UNIVERSITY OF GENOVA

POLYTECHNIC SCHOOL

DIME

Department of Mechanical, Energy, Management
and Transportation Engineering

Master of Science Thesis
in
Mechanical Engineering

Open-source Shape Optimization: an application to Bulbous Bow

Supervisor:

Chiar.^{mo} Prof. Ing. Jan Oscar Pralits

Co Supervisor:

Dott. Ing. Joel Guerrero

Candidate:

Alberto Cominetti

March 2017

Open-source Shape Optimization: an application to Bulbous Bow

Alberto Cominetti

supervised by
Prof. Ing. Jan Oscar Pralits
&
Dott. Ing. Joel Guerrero

Abstract

Optimizing the shape of a given geometry can be a very expensive task, especially in the experimental field. This can be overcome by the introduction of numerical simulations; many commerical software can be used effectively. However this kind of problem often requires the interaction of various codes, then the acquisition of licenses could become unaffordable. So, focusing on fluid dynamics, an open-source shape optimization framework is developed here, involving the combined use of OpenFOAM, Dakota and MiMMO library. The procedure is tested on a practical naval case, i.e. the optimization of the bulb's shape. In particular the effect of protrusion and immersion of this ship's component is investigated through Surrogate Based Optimization. In addition benchmarks of the OpenFOAM numerical model and the effect of sailing speed on the response surface are presented.

Ottimizzazione di forma tramite software Open-source: applicazione ad un bulbo navale

Alberto Cominetti

supervisionato da
Prof. Ing. Jan Oscar Pralits
e
Dott. Ing. Joel Guerrero

Sommario

Ottimizzare la forma di una data geometria può rivelarsi un compito dispendioso, specialmente se l'indagine è condotta sperimentalmente. Tale difficoltà può essere sormontata con l'utilizzo di metodi numerici; in tal senso numerosi software commerciali possono essere impiegati. Tuttavia, l'utilizzo combinato di diversi codici, tipico di queste analisi, potrebbe portare ad elevati costi di licenze . L'obiettivo del lavoro di ricerca qui presentato è stato lo sviluppo di un framework basato su codici open-source. Concepito prinicipalmente per lo studio degli aspetti fluidodinamici, tale struttura prevede l'utilizzo combinato di OpenFOAM, Dakota e della libreria MiMMO. La sua efficacia è stata testata in un caso applicativo del settore navale, ovvero l'ottimizzazione del bulbo. In particolare sono stati studiati gli effetti dell'allungamento e dell'immersione di tale componente sulla resistenza idrodinamica dello scafo. Ciò è stato fatto usando un approccio basato sulla costruzione di modelli surrogati. Inoltre sono state effettuate due validazioni del modello numerico sviluppato in OpenFOAM e uno studio dell'effetto della velocità sulle superfici di risposta.

Acknowledgements

I would first like to thank my supervisor Prof. Jan Pralits, who has provided me all the elements necessary to develop the project, caring about everything, keeping the work organized and giving me a constant support, and my co-supervisor Dr. Joel Guerrero for the great amount of knowledge and passion that has transmitted me and for its continuous and effective support.

This research work has involved many realities, so I'd like to thank all the experts that have give me tips and guidances, among them: Diego Villa, Rocco Arpa, Edoardo Lombardi, Luigi Grossi and Paolo Tornese.

A big thanks goes also to the Flubio Lab, in particular to the PhD students Edoardo and Stefano.

My biggest thanks goes to my mother, my father and my sister for the moral and economical support during this years and for having always sustained and accepted my choices.

I thanks also Gio and Andre, my partners in this long and hard academic path, alone it would have been much more difficult.

However my university career has been also very funny and for that I'd like to thank all Cattaneo's housemates: Emanuele, Umbe, Giulia, Giulio, Gigi, Niko, Fra, Ola, Pietro and all the "acquired" ones.

A big thanks goes to all my friends of Soka Gakkai and especially the "Molo" group for the atmosphere of joy and the precious encouragements they constantly give me.

I'd like to thanks also mestre Rocco and all the angoleiros for the warm and friendly atmosphere I have found at the Saravà.

Introduction

Optimizing the shape of a given geometry can be a very demanding task, especially in the experimental field, where economical constraints usually limit the number of geometries to be tested. Moreover, it is required an in-depth knowledge of the problem being studied in order to choose which are the more appropriate configurations to analyze.

In this view, the continuous development in numerical methods and computing performance has suggested the introduction of software able to model and simulate the case, avoiding the construction of prototypes. These toolkits provide an high amount of features, allowing very detailed analysis, but their use often requires the acquisition of expensive licenses.

Parallel to the world of commercial software there are Free Open Source software. This category refers to the GNU licenses, which, besides the free use, allows the freedom to read, write and redistribute the source code of the applications.

In this context raises the purpose of the work presented in this thesis, i.e. the creation of an Open Source framework to effectively deal with shape optimization, in particular focusing on fluid dynamics problems. In doing so, essentially three tasks have to be addressed:

- how to simulate the *physical behavior*
- how to create the *geometries*
- how to search the *optimal configurations*

The efficacy of the framework has been tested on a practical naval application. Thanks to the data supplied by *Fincantieri S.p.A.*, it has been possible to perform the optimization study of a ship's bulb. Furthermore, the availability of results from experimental tests has allowed a benchmark of the numerical model.

The effect of the bulb is to reduce the wave-making resistance, which is one of the components of the total hydrodynamic resistance of the hull. In the literature

[20] this component is usually described by using six nondimensional parameters. In this application the effects of only two of these have been studied: one representing the protrusion of the bulb and the other representing its immersion.

To simulate the physical behavior of the hull Computational Fluid Dynamics (CFD) has been used. The open source tool employed to compute the hydrodynamic resistance is OpenFOAM. The numerical set up of this simulation is a trade off between accuracy (with respect to experimental trends) and computational resources (memory, clock wall time, storage).

Regarding the generation of geometries the shape morphing approach is used; this allows the creation of new geometries by deforming the original one, avoiding to deal with the entire geometry generation process. This task is accomplished by a shape morpher built using the blocks provided with the MiMMO library.

The poor knowledge of the case study and the inevitable costs of CFD simulations have led to the choice of surrogate modeling as optimization strategy. This consists in the construction of an inexpensive low-fidelity model able to reproduce the outputs of the expensive CFD model. To do this a certain number of configurations has to be simulated with the high fidelity model and these are defined by Dakota toolkit. Then this software is also used to construct the surrogate model.

Putting together all the "ingredients", the optimization procedure can be summarized as follows: Dakota defines a certain number of geometries that are obtained with the shape morpher, built with the MiMMO library, and then simulated by an OpenFOAM numerical model; finally the simulated resistance values are used by Dakota to construct a surrogate model able to inexpensively predict the effect of the two parameters on ship's resistance. To find the optimum is just a matter of searching across a surface.

This thesis is structured in various parts, each one dealing with an aspect of the optimization framework. *Part I* gives an overview of the open source philosophy, with a description of the tools used. In *Part II* the case study is presented, with a brief introduction to the main naval concepts involved. *Part III* is dedicated to the numerical model set up (assumptions, domain definition, boundary conditions, mesh, turbulence etc.). *Part IV* deals with the shape morphing approach, introducing the theory and showing the working principle of the shape morpher. In *Part V* the main aspects of optimization are presented, with a description of the common methods and a look at two surrogate modeling techniques (polynomial regression and Kriging interpolation). Finally, in *Part VI* the results obtained are presented.

Contents

I Open Source Tools 1

1 Free Open Source Software Philosophy 2

2 OpenFOAM 5

2.1 What is OpenFOAM? 5

2.2 OpenFOAM case structure 7

3 MiMMO/MIMIC 9

3.1 What is MiMMO/MIMIC? 9

4 Dakota 11

4.1 What is DAKOTA? 11

4.2 DAKOTA input file 13

II Case study 15

5 Naval concepts 16

5.1 Naval terminology 16

5.2 Resistance Decomposition 18

5.3 Dimensional Analysis 20

6 What is a bulb? 22

6.1 History 22

6.2 Basic principles 24

III OpenFOAM Experiment Set Up 27

7 Assumptions and Equations 28

8 Domain & Boundary Conditions **33**
8.1 Symmetry: half a problem 33
8.2 Reference system 33
8.3 Dimensions 34
8.4 Boundary conditions 35

9 Meshing **38**
9.1 Meshing tools 39
9.2 Coarse & Fine Meshes 40
9.2.1 Background mesh 41
9.2.2 Free Surface refinement 41
9.2.3 Near-Hull refinement 43
9.2.4 Generating the Mesh 43
9.2.5 Checking quality 45
9.3 Remarks 46

10 Temporal Discretization **47**
10.1 Temporal discretization 47
10.2 Local Time Stepping 48

11 Turbulence modeling **50**
11.1 Reynolds Averaged Navier-Stokes Equations 50
11.2 Shear Stress Transport $k - \omega$ Model 51
11.3 Wall functions 54
11.4 Remarks 55

IV Shape Morphing **57**

12 Shape Morphing Theory **58**

13 Shape Morphing Set up **60**
13.1 Definition of Code Blocks 61
13.2 Execution Chain 63
13.3 Input Parametrization 64

V Optimization 66

14 Introduction to Optimization 67

14.1 Mathematical description 67

14.2 Classification of optimization methods 68

14.3 How to choose an optimization method? 70

15 Surrogate Modeling 72

15.1 Key stages 73

15.1.1 Screening 73

15.1.2 Observing 73

15.1.3 Learning 74

15.1.4 Facing the noise 74

15.2 Polynomial & Kriging 74

15.2.1 Least-squares Polynomial Regression 75

15.2.2 Kriging Interpolation 76

VI Results 80

16 Numerical Model Benchmarking 81

16.1 Catching the Resistance-Velocity Trend 82

16.2 Numerical dependencies 83

17 One Design Variable Optimization: protruding the bulb 87

17.1 Catching the trend 88

17.2 Evaluating the noise 91

17.3 Slightly modifying the schemes 92

17.4 Conclusions 94

18 Two Design Variable Optimization: displacing the nose 99

18.1 Displacements and Kracht Parameters 99

18.2 Building the Surrogates 101

18.3 Testing the surrogates 104

18.4 Infilling and Comparing 108

18.5 Validating 110

19 An approach to robust optimal solution 113

Conclusions and Future Developments **119**

Part I

Open Source Tools

Chapter 1

Free Open Source Software Philosophy

One of the main keys of this research work is the combined use of three *Free Open Source software*. These are:

- OpenFOAM: to simulate the physics of the problem by using a finite volume approach;
- MiMMO/MIMIC: to morph the geometry being studied;
- Dakota: to manage the optimization loop, i.e. parametrization, shape morphing, running simulations, creating surrogates, search minimum etc.

All these software are distributed under the GNU General Public License (GPL) or under the GNU Lesser General Public License (LGPL) [1].

Before describing the capalibities of these powerful tools, it is useful to give a brief overview of the ideas behind the *GNU Project*. This is not meant to be an historical overview of the free software movement, but just an introduction to free software's world and to the concepts of GNU licenses.

The initial announcement of the GNU project was made in September 1983 by Richard Stallman and confirmed later with the GNU manifesto (1985). This movement born against the non-free software's dominancy with the purpose of developing an efficient and complete free operating system.

As declared on GNU's website [1], a software can be called free if it allows four essential freedoms:

[1]The difference between these two license concerns the interaction with *proprietary software*.

- The freedom to run the program as you wish, for any purpose.
- The freedom to study how the program works, and change it so it does your computing as you wish (open-source).
- The freedom to redistribute copies so you can help your neighbor.
- The freedom to distribute copies of your modified versions to others, giving the whole community a chance to benefit from your changes.

Practically a free software must be released under a free software license; usually the *GNU General Public License (GPL)* is used, but there are also other GNU licenses, for example the *GNU Lesser General Public License (LGPL).* Regarding the tools used in this research work, *OpenFOAM* is licensed under the former, while *Dakota* and *MiMMO/MIMIC* are licensed under the latter.

Without entering the debate between free and proprietary software, the preamble of the *GNU General Public License* is presented to explain the possibility that this category of software gives.

> *"The GNU General Public License is a free, copyleft license for software and other kinds of works.*
>
> *The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program - to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.*
>
> *When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.*
>
> *To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain*

responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users. Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free." [16]

Chapter 2

OpenFOAM

2.1 What is OpenFOAM?

OpenFOAM® (*Open source Field Operation And Manipulation*) is essentially a C++ library to solve Ordinary and Partial Differential Equations.

Its primary usage is the creation of executables (*applications*) to solve specific problems (*solvers*) or to manipulate data (*utilities*). In particular it has an important usage in solving continuum mechanics problems, including *Computational Fluid Dynamics (CFD)* based on the *Finite Volume Method (FVM)*. OpenFOAM® is supplied with pre- and post-processing environments, its overall structure is shown in *Figure 2.1*.

OpenFOAM® has an extensive range of features to simulate anything from turbulent flows in automotive aerodynamics, to fires and fire suppression in buildings, involving combustion, chemical reactions, heat transfer, liquid sprays and films. It includes tools for meshing in and around complex geometries (e.g. a vehicle), and for data processing and visualisation, and more. Almost all computations can be executed in parallel as standard to take full advantage of today's multi-core proces-

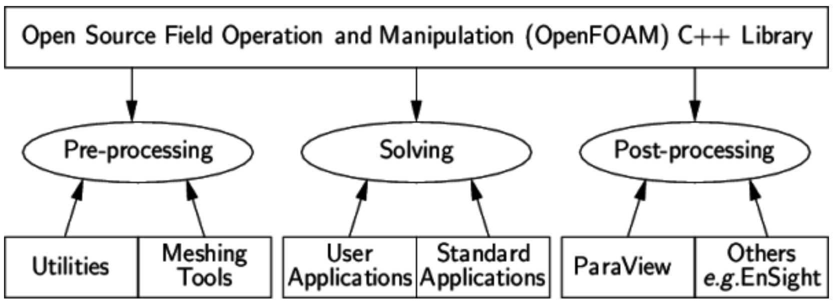


Figure 2.1: *Schematization of* OpenFOAM® *structure. (from CFD Direct website)*

sors and multi-processor computers. On the OpenFOAM® official website some of the available features are listed [2]:

- Fluid Dynamics & Physical Modeling
  - Turbulence modeling (Reynolds-Averaged (RAS), Large-Eddy Simulation (LES), Detached-Eddy Simulation (DES,DDES,etc)
  - Thermophysical modeling
  - Transport/rheology
  - Multiphase flows
  - Rotating flows with multiple reference frames (MRF)
  - Rotating flows with arbitrary mesh interface (AMI)
  - Dynamic meshes
  - Compressible/thermal flows
  - Conjugate heat transfer
  - Porous media
  - Lagrangian particle tracking
  - Reaction kinetics/chemistry
- Geometry & Meshing
  - Mesh generation for complex geometries with *snappyHexMesh*
  - Mesh generation for simple geometries with *blockMesh*
  - Mesh conversion tools
  - Mesh manipulation tools
- Numerical Solution
  - Numerical method
  - Linear system solvers
  - Ordinary Differential Equation system solvers
- Computing & Programming
  - Equation syntax
  - Libraries of functionality

  - Parallel computing

- Data Analysis

  - *ParaView* post-processing
  - Post-processing command line interface (CLI)
  - Graphs and data monitoring

Furthermore OpenFOAM® is a free and open source software under the *GNU General Public License* (see *Chpater 1*), allowing the creation of new solvers and utilities by its users and the presence of a wide-spread community that contributes actively to its continuous development.

2.2 OpenFOAM case structure

The basic directory structure for a OpenFOAM® case, with the minimum set of files required to run an application, is presented in *Figure 2.2.*

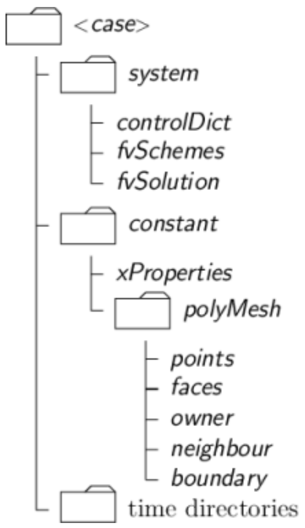


Figure 2.2: *Structure of an* OpenFOAM® *case. (from CFD Direct website)*

The roles of the main directories, contained in the case folder, are listed below:

- `system`, it contains the dictionaries to set up the entire solution procedure (from meshing to solving); at least it must contain three files:

  - `fvSchemes` to specify (run-time) the numerical schemes to discretize the equations;

  - `fvSolution` to set equation solvers, tolerances and other algorithm controls;
  - `controlDict` to control (run-time) the simulation run (start/end time, time-step, function objects etc.)

- `constant`, it contains a folder (`polyMesh`) with the full description of the case mesh and files that specify the physical properties involved (transport and turbulence properties, gravity, dynamic properties etc.)
- `time directories`, it contains files that represent the specific fields at initial condition (e.g. `0` folder) or computed by OpenFOAM® (e.g. `0.01`, `0.02`, ... folders[1]) at consecutive times; it must be underlined that OpenFOAM® always require fields to be initialized, even in steady-state problems

A lot of pages should be written to exhaustively explain OpenFOAM®, but that is beyond the scope of this thesis. However, since this software is at the base of the optimization procedure developed here, the structure and the main files involved in the simulation have been briefly presented.

For further details the *CFD direct* website [2] is suggested.

[1]The name of the folder corresponds to the simulated time at which data are written.

Chapter 3

MiMMO/MIMIC

3.1 What is MiMMO/MIMIC?

MIMIC is a commercial software developed and distributed by *OPTIMAD Engineering srl.* It is designed for dealing with surface and volume mesh morphing and shape optimization in industrial applications. The code is based on MiMMO (*Manipulation and Morphing for Meshes Open Source*, that is an open source C++ library for manipulation and morphing of surface and volume meshes. This too is developed by *OPTIMAD Engineering srl* and licensed under the *GNU Lesser General Public License (LGPL).* MiMMO, in turn, is based on another open source C++ library called BITPIT, which supplies the basic tools to manage surface and volume meshes.

The package is organized as a library of blocks that can be linked and organized in an execution chain, like in a block diagram workflow. It offers a large set of blocks aimed to perform basic or advanced morphing of surface or volume meshes. Essentially two big families of manipulators are implemented:

- *extended-Free Form Deformation (eFFD);*
- *Radial Basis Functions (RBF) morphing techniques.*

It provides also additional features to manage the deformed object, such as user-defined sub-selections of the target geometry, penetration checks of deformed mesh against external object and wrappers to parametric manipulators for more intuitive basic deformations [3]. Furthermore, it can interface with the most common geometrical formats (.stl, .vtu, .nas and others).

MIMIC, compared to MiMMO, provides advanced tools and interfaces, such as volume mesh morphing, OpenFOAM input/output interfaces and management of continuity constraints by means of Level-Set techniques.

All the MiMMO objects to use for a specific case can be pre-defined and then inserted in a container, the *execution chain*. This allows the automatic ordering of these objects and the managing of their relationship, leading to a more straightforward set up and execution of the workflow.

This has been just an overview of MiMMO/MIMIC capabilities; further info on MiMMO/MIMIC are available on the *OPTIMAD* official website (`http://www.optimad.it/`).

A look at the theory behind the approach used here and at the execution chain built for this research work will be given in *Part IV*.

Chapter 4

Dakota

4.1 What is Dakota?

Dakota (*Design and Analysis toolKit for Optimization and Terascale Applications*) is a general-purpose software-toolkit for performing optimization, uncertainty quantification, parameter estimation, design of experiments and sensitivity analysis on supercomputers.

This C++-based toolkit provides an extensible and flexible interface between analysis codes and iterative system analysis methods [4]. Dakota delivers a wide variety of iterative methods and meta-algorithms that in user's manual are classified as follows [12]:

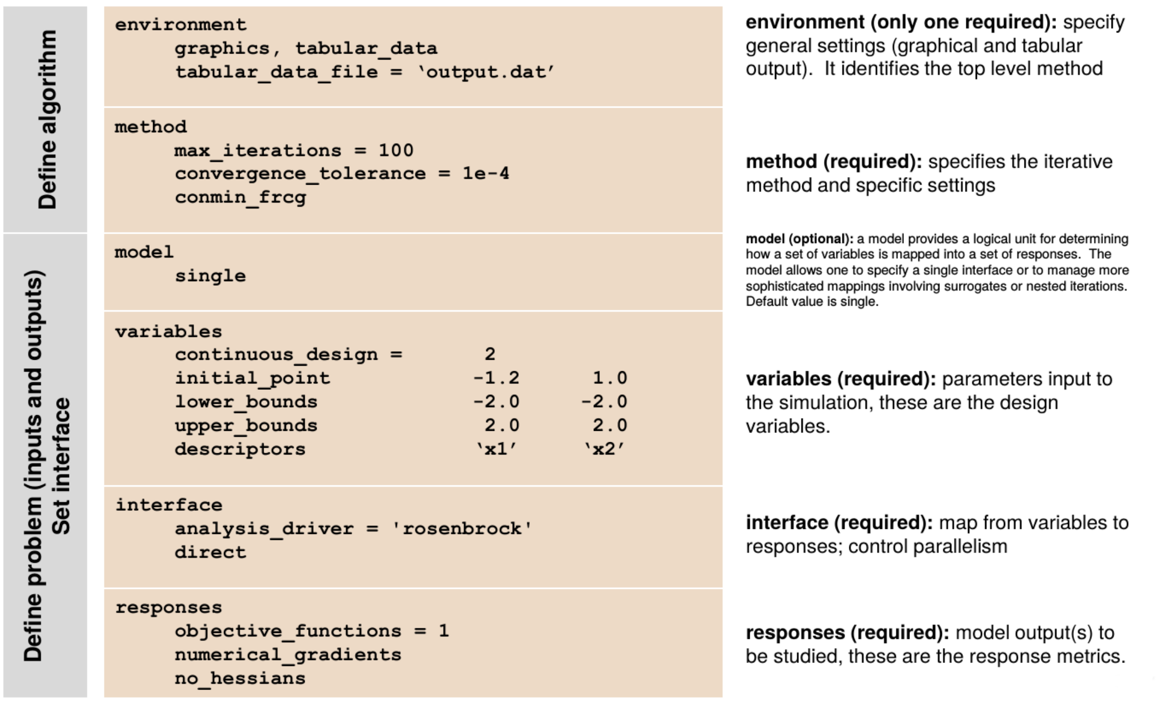
- Parameter Studies: employ deterministic designs to explore the effect of parametric changes within the simulation code, yielding one form of sensitivity analysis. Exploration is done by a deterministic selection of points (structured or specified by user) in each of the four available type of parametric studies:
  - *vector*, perform a parametric study along a line between two points defined in a n-dimensional parameter space, where the user define the number of steps to use;
  - *list*, the user supplies a list of points of a n-dimensional parameter space where the response is evaluated by the specified simulation code;
  - *centered*, given a point in a n-dimensional parameter space, nearby points along the space axes are evaluated;
  - *multidimensional*, creates a regular lattice or hypergrid in a n-dimensional parameter space.

- Design of Experiments: employ design and analysis of computer experiments (DACE) techniques to explore the parameter space of an engineering design problem; similar to Parametric Studies, but the primary goal is to generate a good coverage of the input parameter space, extracting as much trend data as possible using a limited number of sample points.
- Uncertainty Quantification: determination of effect of input uncertainties and assumptions on model outputs or results. This is done by characterizing input uncertainties, forward propagating them through a computational model and performing statistical or interval assessments on the resulting responses. It is related to Sensitivity Analysis but has a more quantitative approach by using some uncertainty structures (e.g. probability distributions). Dakota provides methods for both epistemic and aleatory uncertainties.
- Optimization: minimization (or maximization) of an objective function, typically calculated by the user simulation code, subjects to constraints on design variables and responses. Available methods can be classified as follows:
  - *Gradient-Based Local Methods*
  - *Derivative-Free Local Methods*
  - *Derivative-Free Global Methods*
  - *Multiobjective Optimization*
  - *Automatic Scaling*
- Calibration: seek to maximize agreement between simulation outputs and experimental data (or desired outputs).
- Surrogate Modeling: creation of inexpensive approximate models that capture the salient features of an expensive high-fidelity model. Dakota provides several data fit methods to construct surrogates, but has also multifidelity and reduced-order models.

Furthermore, Dakota permits the creation of *Nested Models* by layering on method over another.

Before looking at the Dakota case structure, let's have a look at its important advantages, motivating why it is convenient to use this toolkit.

First of all, it is licensed under the *GNU Lesser General Public License (LGPL)* (see *Chapter 1*), hence it is free and open source. From the computational point of view, the main reasons for using it are [17]:



```
environment
     graphics, tabular_data
     tabular_data_file = 'output.dat'

method
     max_iterations = 100
     convergence_tolerance = 1e-4
     conmin_frcg

model
     single

variables
     continuous_design =        2
     initial_point            -1.2      1.0
     lower_bounds             -2.0     -2.0
     upper_bounds              2.0      2.0
     descriptors              'x1'     'x2'

interface
     analysis_driver = 'rosenbrock'
     direct

responses
     objective_functions = 1
     numerical_gradients
     no_hessians
```

environment (only one required): specify general settings (graphical and tabular output). It identifies the top level method

method (required): specifies the iterative method and specific settings

model (optional): a model provides a logical unit for determining how a set of variables is mapped into a set of responses. The model allows one to specify a single interface or to manage more sophisticated mappings involving surrogates or nested iterations. Default value is single.

variables (required): parameters input to the simulation, these are the design variables.

interface (required): map from variables to responses; control parallelism

responses (required): model output(s) to be studied, these are the response metrics.

Figure 4.1: *Example of* Dakota *input file (Courtesy of Wolf Dynamics srl)*

- ability to interface with a generic black-box solver;
- scalable parallel computation from desktops to clusters;
- its capabilities are extensively validated;
- fully scriptable;
- ability to capture simulation failures;
- restart capabilities;
- parallel asynchronous or concurrent evaluations;

4.2 Dakota input file

Dakota take as input only one file that contains all the problem information grouped in six blocks: `variables`, `interface`, `responses`, `model`, `method` and `environment`. An example of Dakota input file is shown in *Figure 4.1*.

There is an inherent relation that ties this boxes together, for almost Dakota analysis it can be summarized as follows: in each iteration of its algorithm, a `method` block requests a `variables-to-responses` mapping from its mode, which the `model`

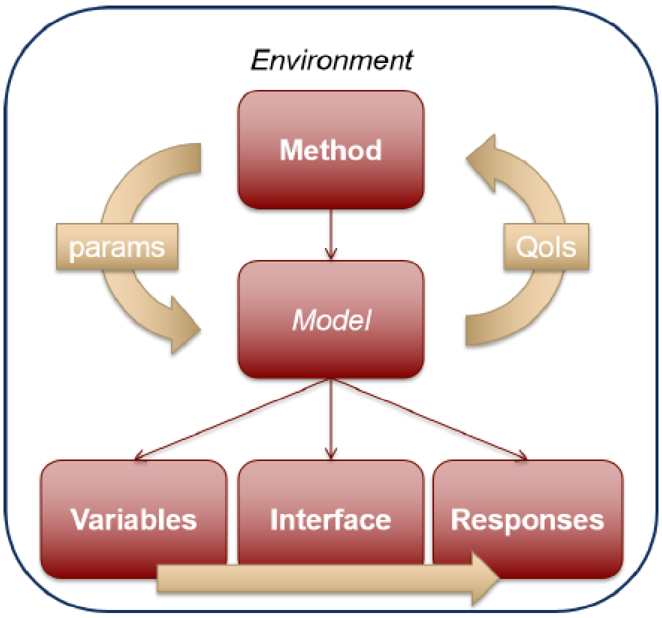


Figure 4.2: *Schematization of the common* DAKOTA *input file blocks relationship (from Dakota v6.5 User's Manual [12])*

fulfills through an `interface`. An example of this basic relationship is schematized in *Figure 4.2*. However, more advanced cases are possible [12].

An in-depth description of DAKOTA toolkit is beyond the scope of this introductory chapter. The aim has been to give a general introduction to its features and capabilities. Among all of these only some are used in this research work and the theory behind them is presented in *Part V*.

For further details on DAKOTA toolkit the official website [4] is recommended.

Part II

Case study

Chapter 5

Naval concepts

The main subject of this thesis is the development of an optimization strategy. The procedure wants to be general, but to give it a sense of application a case study has been chosen. The choice has fallen on a naval problem, in particular the shape optimization of the ship's bulb. As will be presented in this chapter, this component leads to a reduction of the hydrodynamic resistance and, as it is explained in literature, its efficacy depends on its shape. Hence this is an interesting example of shape optimization.

Hence before dealing with the numerical methods, an overview of some naval concepts is necessary. In this chapter the terminology used to identify the hull, the concepts of resistance and scaling law and a description of the bulb are presented.

5.1 Naval terminology

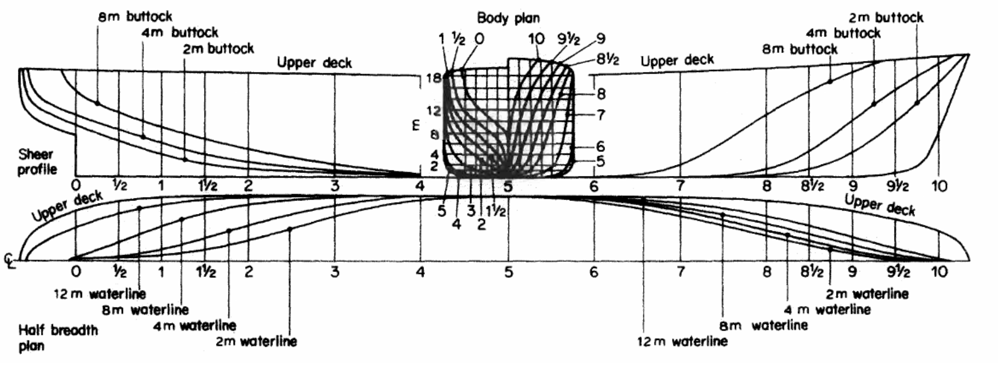
In the following chapters some naval terms are used to describe the hull, so the naval terminology adopted is reported [25].

- The hull shape can be described by intersecting it with three mutual orthogonal planes:
  - horizontal planes, called *waterplanes*, that are parallel to the calm water free surface and cut the hull in *waterlines*;
  - longitudinal planes that cut the hull in *buttock lines* (*Figure 5.1.a*);
  - transverse planes that cut the hull in *transverse sections*;

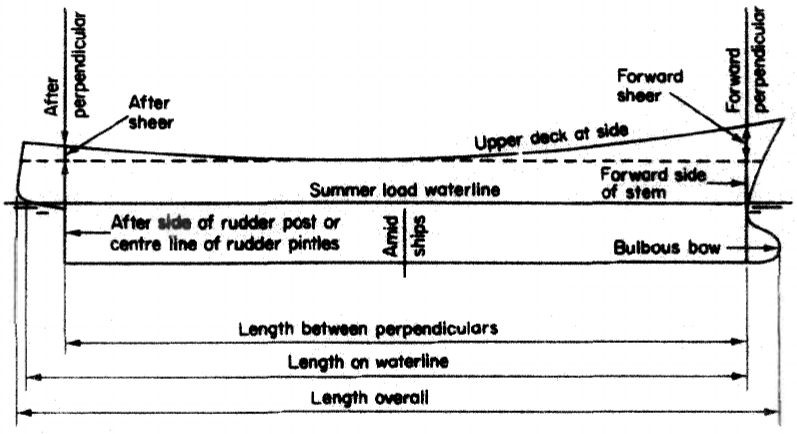
  Usually the hull is symmetric about the longitudinal plane and the buttock corresponding to the symmetry plane is called *profile.*

- *Design waterline* is the waterline at which the ship floats in the design condition (commonly at the maximum available load).
- *Fore perpendicular (FP)* is a line perpendicular to the load line and passing through the intersection of the forward side of the stem with the design waterline.
- *Aft perpendicular (AP)* is a line perpendicular to the design line and passing through the centreline of the rudder pintle or the after side of the rudder post.
- *Length between perpendiculars (L_{PP}),* is the distance measured along the load waterline between the after and fore perpendiculars.
- *Midship section* is the transverse section locate *amidships*, i.e. at the mid-point between the perpendiculars.
- *Beam (B)* is usually considered as the width of the hull at amidships.
- *Draught (T)* is the vertical distance of the lower points of the hull from the water level.
- *Trim* is the difference between the draughts at forward and aft; it depends mainly to the load condition in junction to the hull shape. The dynamic one is related to the attitude of the ship due to its speed or motion.
- *Sinkage* is a ship's motion that lead to a variation of the ship's draught.
- *Displacement* is the weight of water displaced by the ship, i.e. the weight of its underwater volume (Archimede's principle). It can be expressed in *Newton*, in m^3 (displacement volume) or in *kg* (displacement mass).

Without entering into the details of ship's design, it must be highlighted that the *displacement volume* and the *length between perpendiculars* are some of the parameters that characterize the main hull. This is taken into account during shape morphing, ensuring that the bulb's deformations don't vary them significantly. Even if a strictly bulb's design procedure is beyond the purpose of this work, it has been considered worthwhile to evaluate the bulb effects without significantly changing the ship parameters.



(a) *Buttocks and waterlines.*



(b) *Hull parameters.*

Figure 5.1: *Representation of some of the terminology used in naval architecture. (from "Introduction to Naval Architecture" by E. Tupper [25])*

5.2 Resistance Decomposition

The resistance is the force to move the ship at the desired speed. Often it is referred to smooth water and, when appendages are excluded, it is called *bare-hull resistance*.

Calculation of the ship's resistance are important in order to design the adequate propulsive system. Unfortunately this is not an easy task, since the total resistance depends on a lot of factors interacting one with each other in an extremely complicated way [18]. To overcome these difficulties it is common practice to decompose the ship resistance in different components.

Looking at a ship moving through the water, two main physical phenomena comes to the eye: waves and turbulence. These correspond to a distribution of pressure and shear forces over the hull. Hence, by considering the *forces acting*, a first physical breakdown of resistance could be made [10]:

- *frictional resistance*, it is the sum of all tangential shear stresses τ acting on the hull and hence it is caused only by the viscosity of the fluid;
- *pressure resistance*, it is the sum of all pressure forces p acting on the hull and hence it depends both on viscosity and waves.

Conversely, looking at the *energy dissipation*, another decomposition can be made:

- *viscous resistance,* it is the sum of all forces caused by the presence of viscosity, i.e. friction and wake;
- *wave-making resistance,* it represents the energy drained from the body to sustain the hull wave system.

These two decomposition are visualized in *Figure 5.2.*

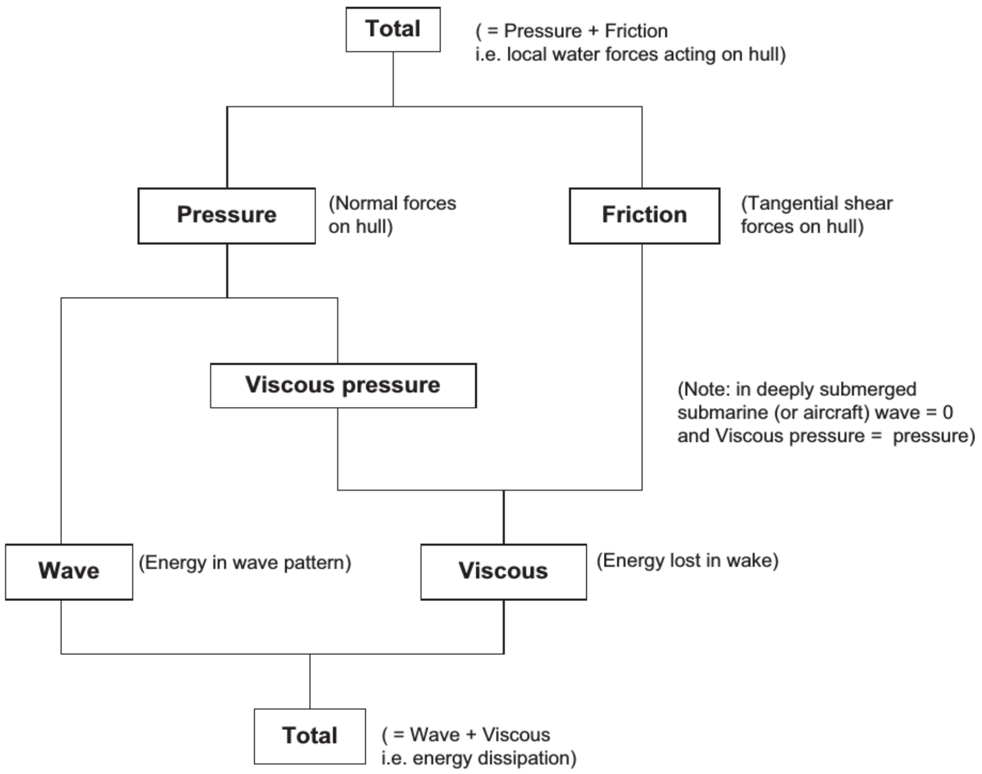


Figure 5.2: *Basic hull resistance decomposition (from "Ship Resistance and Propulsion" by F.Molland et al. [10])*

This is just a basic decomposition, there are other physical phenomena that should be considered like spray and wave-breaking, especially at high sailing speeds. However this resistance decomposition is very useful in experimental predictions.

5.3 Dimensional Analysis

Much of the knowledge about ship resistance has been learned from towing tank tests and it is impossible to speak of its components without referring to model scale [18]. For this reason the main concepts behind this approach are presented here.

In order to extract some qualitative information about the physics a dimensional analysis can be carried out. This approach constitutes the base of model experiments.

First, let's define the main quantities that affect ship's resistance [18]:

- *speed V*
- *size L*
- *density ρ*
- *dynamic viscosity μ*
- *gravity g*
- *pressure p*

Applying the dimensional analysis, a non-dimensional relationship can be obtained:

$$C_T = \frac{R}{0.5\rho S V^2} = f(\frac{VL}{\nu}, \frac{gL}{V^2}, \frac{p}{\rho V^2}) \tag{5.1}$$

where $\nu = \frac{\mu}{\rho}$ is the *kinematic viscosity*, $S \propto L^2$ is the *wetted surface.*

The power of equation (5.1) is that it states that geometrically similar body of different size with the same values for all the right-hand size terms will have the same resistance coefficient $(\frac{R}{0.5\rho S V^2})$.

Since the third term $\frac{p}{\rho V^2}$ can be neglected (if there is no cavitation), the above relationship could be rearranged by recovering two nondimensional parameter: *Froude number $Fn = \frac{V}{\sqrt{gL}}$* and *Reynolds Number $Re = \frac{VL}{\nu}$*.

$$C_T = f(Re, Fn) \tag{5.2}$$

To allow the study of the full ship at sizes that can be tested in towing tank, both *Froude* and *Reynolds* similitude have to be respected. Unfortunately this requires that:

$$Re_{model} = Re_{ship} \Longrightarrow \frac{V_{model} L_{model}}{\nu_{model}} = \frac{V_{ship} L_{ship}}{\nu_{ship}} \Longrightarrow V_{model} = V_{ship}\lambda \tag{5.3}$$

$$Fn_{model} = Fn_{ship} \Longrightarrow \frac{V_{model}}{\sqrt{gL_{model}}} = \frac{V_{ship}}{\sqrt{gL_{ship}}} \Longrightarrow V_{model} = \frac{V_{ship}}{\sqrt{\lambda}} \tag{5.4}$$

where $\lambda = \frac{L_{ship}}{L_{model}}$ is the *scale factor* between full scale and model.

Obviously, it is nearly impossible to simultaneously satisfy both requirements (except by using a fluid with different viscosity for the model). Since the wave resistance is a more complex phenomena, the choice fall to the *Froude* approach.

In this context the resistance decomposition becomes important, by saying that if gravity forces are studied properly, as in *Froude similarity*, then *wave resistance* is correctly modeled.

However, in model tests what is measured is the total resistance. So it is common practice to calculate the frictional resistance with empirical formulations and then find the wave resistance by subtraction to the total. For example the ITTC-57 approach uses the following correlation for the frictional coefficient:

$$C_F = \frac{0.075}{(\log_{10} Re - 2)^2} \tag{5.5}$$

An important precaution is to choose a model size that ensures a turbulent flow around the hull, in order not to compare two different flow regimes.

Chapter 6

What is a bulb?

6.1 History

The bulb is a component visible in almost all type of cargo ships.

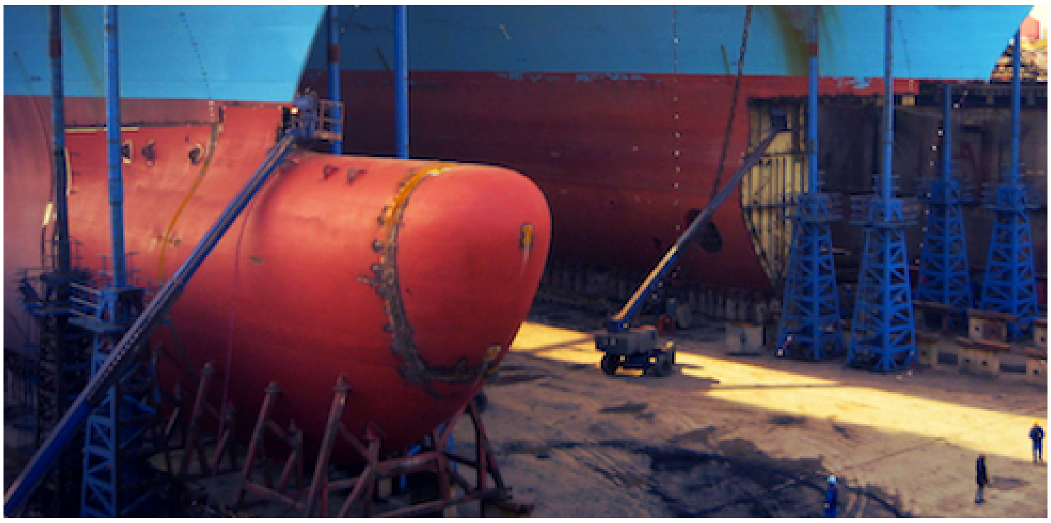


Figure 6.1: *Example of bulbous bow. (courtesy of Maersk Maritime Technology [5])*

Despite its technical developments are relatively recent, bulbous bow has an ancient origin; in fact, it was born as a weapon, the Greek galley's ram. It took many centuries before its fundamental hydrodynamic benefits were discovered.

In 1867 the British civil engineer William Froude revolutionized the study of Naval hydrodynamics by introducing the use of the "towing tank test" to examine at a reduced scale the hydrodynamic of the ship [15].

Going on hundreds test and comparing different models, he caught the distinction of the hull resistance in two components: friction and wave resistance (see *Chapter 5*). Moreover, he noted that the presence of a ram bow seemed to reduce the second one, but attributed this effect to the lengthening of the hull.

Fifty years later, the U.S. naval captain David Taylor and his team equipped the battleship *Delaware* with what it will be called a "Taylor Bulb". After is retirement in 1923 he wrote [15]:

> *In the literature of ship resistance I had seen a reference by Mr. William Froude to a so-called swan model having full but narrow water lines close to the bow [...] the theory seemed to be to have first a false bow as it were, corresponding to a small ship. This would create a small bow wave and in its hollow was located the second or true bow, making a second bow wave that would neutralize the first.*

This is the first time that the bulb's *wave-cancellation effect* has been described. From that moment the scientific development of bulbous bow starts.

In the 30's, the russian naval architects Vladimir I. Yourkevitch proposed the adoption of a "violin hull" in the transatlantic vessel *Normandie*, winning the *Blue Riband*[1] on her maiden voyage with an average speed of 29.98 knots [21]. After this success, newspapers reported [15]:

> *The secret of streamlining, and therefore of speed, is that ships beneath the water will be blunt-bowed.*

In 1940, after having tried over fifty variations of Taylor bulb, Japanese researchers decided to build their new ship *Yamato* with a large protruding bulb, who reduced its resistance by 8 percent [15].

The growth of important mathematical theories for wave description (Mitchell, Havelock, Wigley et al.) leads to change the focus from the hull streamlining to the study of ship's wave pattern. The Japanese researcher Takao Inui developed the "waveless form theory", by focusing on the energy subtracted by the waves.

By the 80s the bulbous bow was being designed and built around the world, and evolved till today into numerous configurations and shapes, but many naval architects still refer to it as "Inui bulb" [15].

Nowaday the bulbous bow is caught in the trade off between fuel efficiency and danger in possible collisions. The regulatory or insurance measures that may be implemented to mitigate the risks are not yet clear [15].

[1] *Blue Riband* is an unofficial accolade to the fastest transatlantic passenger liner.

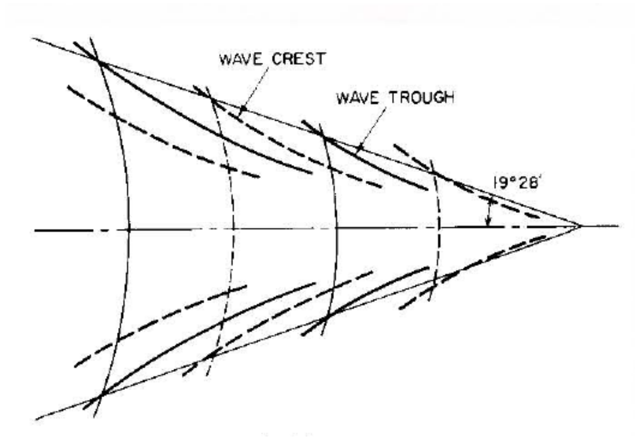


Figure 6.2: *Schematization of the Kelvin wave pattern. (from "Principles of Naval Architecture" [18])*

6.2 Basic principles

To understand the usefulness of the bulb in reducing ship resistance, it is necessary to have some knowledge about wave-making resistance. As it has been already said in the previous chapter, the wave-making resistance is caused by the energy necessary to maintain the hull wave system.

A conceptual representation of the wave system has been given by Lord Kelvin, which considering a pressure point moving on a free surface, noted the presence of two different types of wave: *divergent*, that radiate from the point, and *transverse*, that follow the point. The composition of these two wave system leads to a wave pattern contained within two straight line starting from the pressure point and making an angle of about 19 degrees (*Kelvin angle*) [18].

The ship's wave system presents similar features that can be summarized as follows:

- at the bow there is a large *divergent wave*, followed in diagonal by other divergent waves;
- *transverse waves* starts from the hull with their crest lines perpendicular to the direction of motion till coalescing with the *divergent* ones;
- similar wave systems form at the shoulders and the stern.

At low speeds the waves generated are small and hence also wave resistance is neglectable compared to the viscous component. It's importance will increase with sailing speed, but since also the wave pattern depends on speed, some bump and

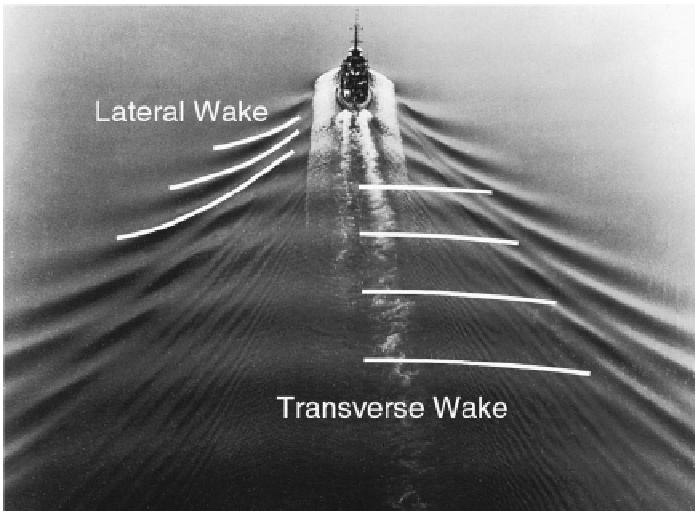


Figure 6.3: *Example of wave pattern generated by a ship. (courtesy of Professor W.H. Munk, Scripps Institute of Oceanography)*

hollows will be present. These are due to the interference of the wave systems, that could be positive, augmenting the resistance, or negative, leading to wave cancellation.

The presence of the bulb creates another wave system that, interfering with the ship's one, leads to a cancellation effects and then reduces the wave-making resistance. It is obvious that its effect can't be seen at low velocities.

The effect of the bulb can be summarized by the words of A. M. Kracht who contributed a lot in the study of bulbous bow [20]:

> *The protruding bulb form affects hydrodynamically a variation of the velocity field in the vicinity of the bow, that is, in the region of the rising ship waves. Primarily the bulb attenuates the bow wave system, which usually is accompanied by a reduction of wave resistance. By smoothing the flow around the forebody, there is good reason to believe that the bulb tends to reduce the viscous resistance too. Therefore, the beneficial action of a protruding bulb depends on the size, the position, and the form of the bulb body.*

In the same paper Kracht introduced six non-dimensional parameter to describe each bulb form:

- Linear
  - *Length parameter C_{LPR}:* protrunding length divided by length between perpendiculars of the ship.

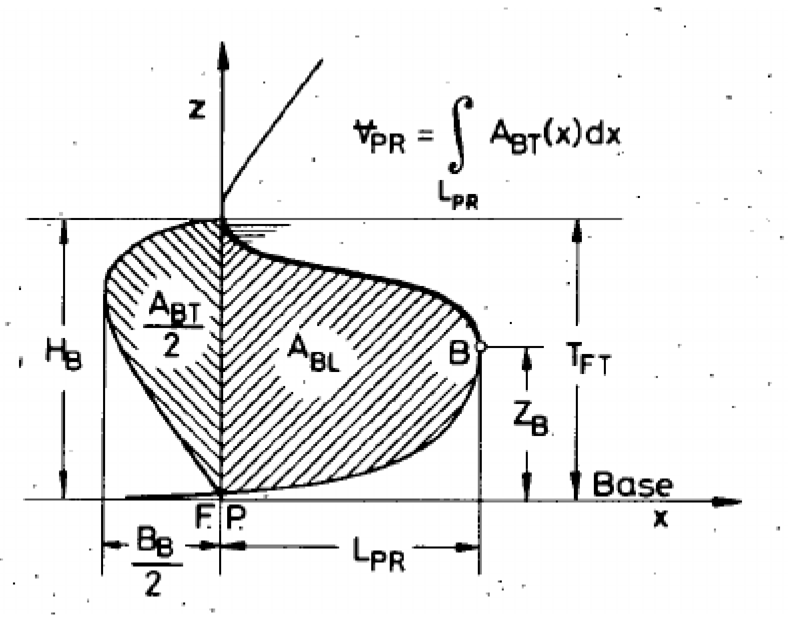


Figure 6.4: *Sketch of a bulbous bow with its main geometrical parameteres. (from "Design of Bulbous Bow" by A.M. Kracht [20])*

  - *Depth parameter C_{ZB}:* heigth of the foremost point of the bulb over the base divided by the draught at *FP*.
  - *Breadth parameter C_{BB}:* maximum breadth at *FP* adimensionalized by the beam of the ship.

- NON-LINEAR
  - *Cross-section parameter C_{ABT}:* cross-sectional at *FP* divided by the midship-section area
  - *Lateral parameter C_{ABL}:* ram bow (after FP) area divided by midship-section area
  - *Volumetric paramter C_{VPR}:* nominal bulb volume adimensionalized by the displacement volume (underwater)

Upon these parameters Kracht constructed a bulb's design procedure that is still used today.

During the optimization procedure developed here, only the first two linear parameters are considered: length and depth parameters.

Part III

OpenFOAM Experiment Set Up

Chapter 7

Assumptions and Equations

The aim of the OpenFOAM numerical model is to simulate the behavior of a ship. In particular, the resistance to its movement has to be predicted. Dealing with fluids means to take on the *Navier-Stokes Equations* system.

Ship resistance prediction involves a series of difficulties like the big dimensions of the geometry, the presence of two phases, the interaction with waves, the presence of turbulence and separation. For this reason, some simplifications need to be introduced in order to lighten the efforts. In doing this, a lot of care must be taken not to corrupt the model. All the assumptions made here follow the common practical guideline and the advice from experts in the naval fields (*Fincantieri Technical Office*, *Distretto Ligure per le Tecnologie Marine*, *Dipartimento di Ingegneria Navale dell'Università di Genova*).

As stated by the *International Towing Tank Committee (ITTC)*, the goal of resistance calculations is the determination of the required power to ensure the ship a certain speed. Since resistance tests in towing tanks are conventionally carried out for a scaled ship model in calm water without propulsor, numerical computations are done to replicate these experiments. Both numerical and experimental results can be extrapolated to full-scale using the *ITTC* procedure [8].

For the ship subject of this research towing tank results are available, thus, to make a comparison possible, the model is built to represent that condition. However, in the numerical model an important simplification is adopted: the *fixed trim condition*. This means that the ship has no degree of freedom, i.e. its position is fixed. This is not the case in towing tank experiments, but, inspecting the experimental results, it has been observed that position changes are neglectable for low velocities, making this simplification admissible.

Advantages that come from this choice are extremely important, because it allows steady calculations, particularly the use of the *Local Time Stepping* technique that, as it will be explained in *Chapter 10*, speeds up a lot the computations.

Moreover, not only the propulsor has been excluded from the computations, but also all other appendages, reducing the geometry only to the hull and ensuring the symmetry about the ship vertical longitudinal plane.

Summarizing, the case study can be imagined like a fixed scaled model of the ship hit by a flow of two separated phases (water and air) at a certain uniform velocity. All the hypothesis are listed below:

- model-scale
- calm water
- no propulsor and no appendages
- fixed trim condition
- symmetry about the vertical longitudinal plane
- constant temperature
- constant fluid (air and water) properties

Phase	**Density $\boldsymbol{\rho}$ $[\frac{kg}{m^3}]$**	**Kinematic Viscosity $\boldsymbol{\nu}$ $[\frac{m^2}{s}]$**
Water	998.3	$1.02 \cdot 10^{-6}$
Air	1	$1.48 \cdot 10^{-5}$

Table 7.1: *Transport properties used in the simulation*

Ship flows are governed by the conservation laws of mass, momentum and energy, grouped in the *Navier-Stokes Equations (NSE)* system.

Reminding that flow around a hull is incompressible and assuming constant temperature, the energy equation becomes useless. Furthermore, air and water follow the Newtonian Fluids relationship between shear stress and strain rate. Thus the *Navier-Stokes Equations* can be written as follows:

$$\begin{cases} \boldsymbol{\nabla} \cdot (\rho \boldsymbol{v}) = 0 \\ \\ \dfrac{\partial \rho \boldsymbol{v}}{\partial t} + \boldsymbol{\nabla} \cdot \{\rho \boldsymbol{v}\boldsymbol{v}\} = -\nabla p + \nabla \cdot \{\mu \left[\nabla \boldsymbol{v} + (\nabla \boldsymbol{v})^T\right]\} + \rho \boldsymbol{g} + \boldsymbol{f}_\sigma \end{cases} \tag{7.1}$$

where:

- $\boldsymbol{v}$ is the *velocity vector*;
- p is the *relative pressure*;
- μ and ρ are the *density* and the *dynamic viscosity* of the fluid considered;
- $\boldsymbol{g}$ is the *gravitational acceleration*;
- $\boldsymbol{f_\sigma}$ is the *surface tension*;

Dealing with multiphase flow, this set of equations isn't enough, fluids interface has to be modeled in some manner. Among the wide variety of multiphase numerical approaches, one of the most used is the *Volume of Fluid (VoF)* method. In particular, the version employed in this work is the one used by the `interFoam` solver.

The principle is to define a quantity called *liquid volume fraction* α that indicates which fluid properties have to be considered:

$$\alpha(\boldsymbol{x}_{cell}, t) = \frac{1}{V_{cell}} \iiint_{cell} I(\boldsymbol{x}, t) dV, \qquad I(\boldsymbol{x}, t) \begin{cases} 1 \ (water) \\ 0 \ (air) \end{cases} \tag{7.2}$$

where $\boldsymbol{x}_{cell}$ is the center of a computational cell where flow variables are computed and V_{cell} its volume. Then fluid properties can be generalized throughout the domain as follow:

$$\mu(\boldsymbol{x_{cell}}, t) = \mu_{water}\alpha(\boldsymbol{x_{cell}}, t) + \mu_{air}(1 - \alpha(\boldsymbol{x_{cell}}, t)) \tag{7.3}$$

$$\rho(\boldsymbol{x_{cell}}, t) = \rho_{water}\alpha(\boldsymbol{x_{cell}}, t) + \rho_{air}(1 - \alpha(\boldsymbol{x_{cell}}, t)) \tag{7.4}$$

where ρ_{water}, ρ_{air}, μ_{water} and μ_{air} are the predefined constant properties of the two phases (*Table 7.1*).

What `interFoam` solver does is computing the *liquid volume fraction* field by solving a modified version of phase transport equation:

$$\frac{\partial \alpha}{\partial t} + \boldsymbol{\nabla} \cdot (\alpha \boldsymbol{v}) + \boldsymbol{\nabla} \cdot (\alpha(\alpha - 1)\boldsymbol{v_r}) = 0 \tag{7.5}$$

$$\boldsymbol{v_r} = min(c_\alpha |\boldsymbol{v}|, max(|\boldsymbol{v}|) \cdot \boldsymbol{n} \tag{7.6}$$

where $\boldsymbol{v_r}$ is the *interface compression velocity*, a trick used to avoid numerical diffusion; coefficient c_{alpha} (`cAlpha` in *OpenFOAM*) is a parameter to control compression and usually set equal to 1 [24].

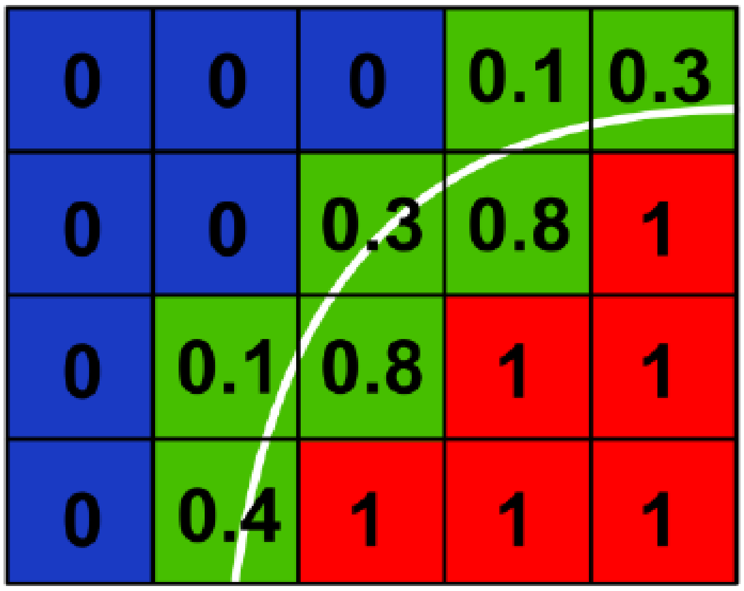


Figure 7.1: *Example of liquid phase fraction field at interface (courtesy of Wolf Dynamics srl)*

Returning to equation (7.1), in this case the *surface tension* is neglected since inertial forces are predominant.

Now that the problem has been mathematically modeled by equations (7.1) and (7.5), *computational fluid dynamic* is used to calculate the flow variables $(\boldsymbol{v}, p, \alpha)$ throughout the domain. This is done with *OpenFOAM* by following the workflow represented in *Figure 7.2*. Each of these steps will be presented in the next chapters of this part.

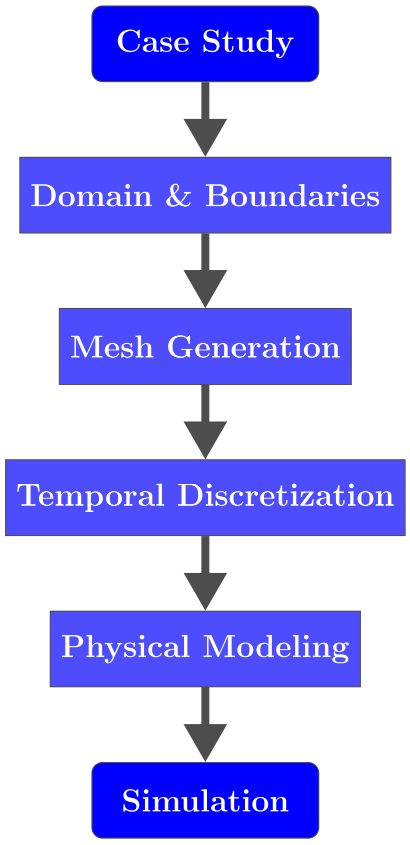


Figure 7.2: *Workflow of the simulation set up*

Chapter 8

Domain & Boundary Conditions

8.1 Symmetry: half a problem

Before starting with the model set-up, an important simplification has to be made.

Since ship geometry is symmetrical about the vertical longitudinal plane and motion is straight in the longitudinal direction, only half of the hull can be modeled without losing information. This assumption takes great advantages in terms of computational costs.

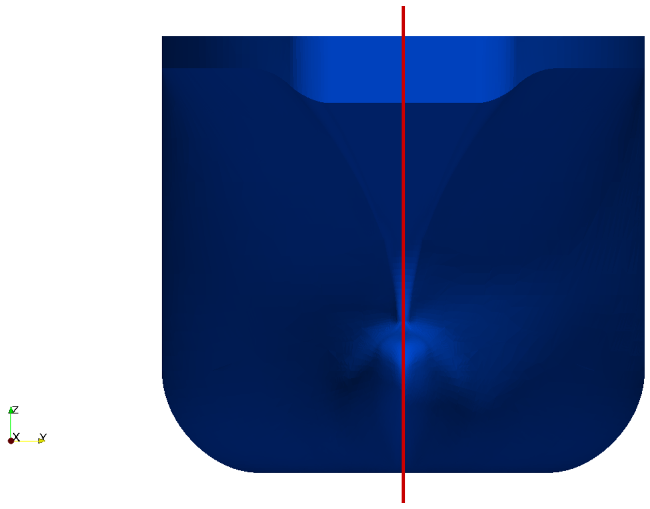


Figure 8.1: *Frontal view of the hull geometry and the symmetry plane (red).*

8.2 Reference system

As in many other practical cases, the reference system is defined as follows:

x-axis start from the *aft perpendicular* and grows positive toward the *fore perpendicular*

y-axis start from the *symmetry plane* and points outward the domain

z-axis start from the *quiet sea level* and grows positive above free surface (right-hand rule)

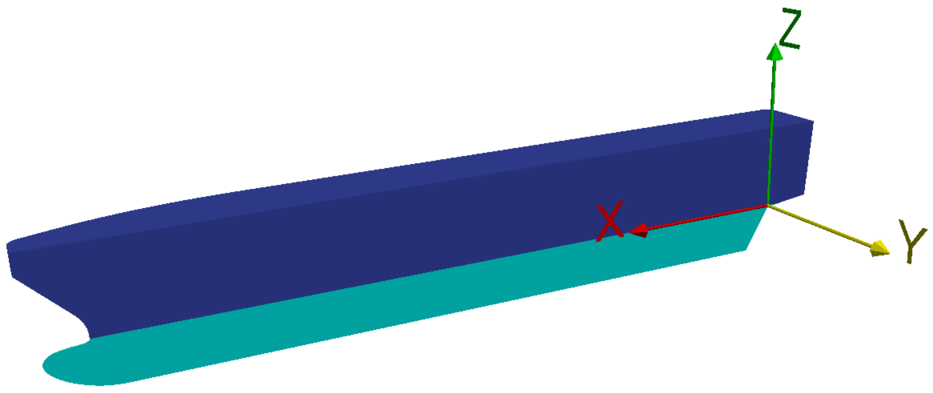


Figure 8.2: *Position and directions of the reference system.*

8.3 Dimensions

The computational domain is a rectangular box to which the hull geometry is subtracted. It is aligned with the reference system and has six faces where flow conditions must be imposed.

The foremost one has been called `inlet` and it is where the flow enters the domain, while the rear patch has been called `outlet`.

Top and bottom patches have been called `atmosphere` and `bottom`.

One side face corresponds to the ship symmetry plane and has been called `midPlane`, while the other one has been simply called `side`.

The main idea behind the choice of domain dimension is to place boundaries *"sufficiently far"* from the hull. This avoids boundary conditions to corrupt the solution.

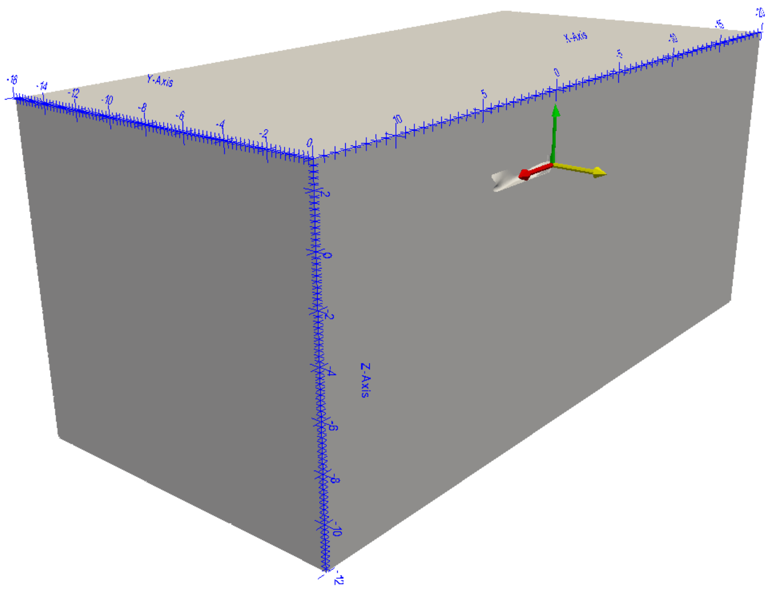


Figure 8.3: *Position and dimension of the computational domain.*

The practical guidelines given by the *26th ITTC Specialist Committee on CFD in Marine Hydrodynamics* suggest to place these boundaries $1 - 2L_{PP}$ away from the hull [7]. It also recommends, in an unsteady case, to put the outlet boundary $3 - 5L_{PP}$ downstream, in order to prevent wave reflection.

These practical values have been exaggerated to be sure that the same domain accurately works also at high velocities.

So inlet and outlet patches are located $2.5L_{PP}$ and $5L_{PP}$ away from bow and transom.

The top and bottom are placed at $0.75L_{PP}$ and $3L_{PP}$ away from quiet sea level.

The side patch is placed at $4L_{PP}$ away from symmetry plane to avoid a lateral reflection of the Kelvin waves.

8.4 Boundary conditions

Boundary conditions are necessary to generate a solution because they tell the solver what's going on at the domain boundaries.

When defining a boundary condition its type, its location and the physical properties it represents are required.

There are several types of boundary conditions available in *OpenFOAM*, the right choice influence accuracy and stability of the simulation.

Looking at the equations presented in the previous chapter, it can be seen that

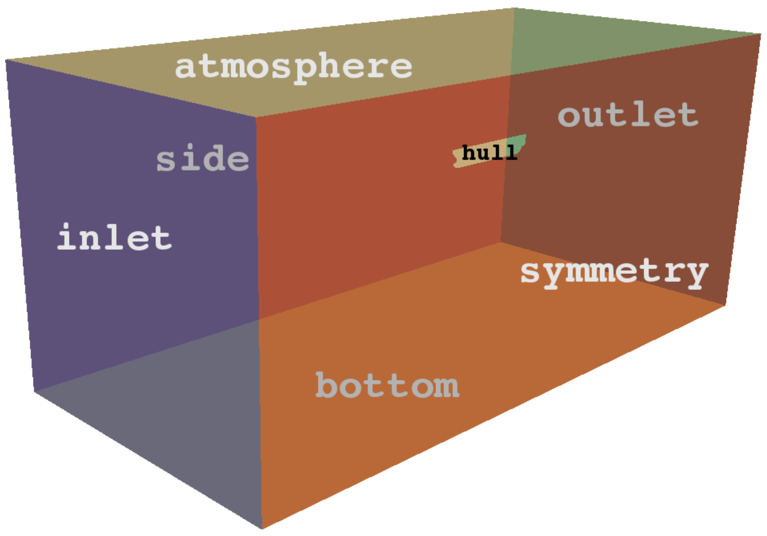


Figure 8.4: *Domain boundaries definition.*

there are five unknown variables: three components of the velocity vector, pressure and phase fraction.

INLET To simulate ship steady motion in calm water, an uniform flux is supposed to enter the domain across the `inlet`.

Velocity : `fixedValue`, assign an uniform value $U = U_\infty$ over all the patch.

Pressure : `fixedFluxPressure`, adjust the pressure gradient to keep the boundary flux specified by the velocity condition.

Phase Fraction : `fixedValue`, assign the value from internal field.

OUTLET Flow at the `outlet` can't be imposed, but it is computed to not influences the physics inside the domain.

Velocity : `outletPhaseMeanVelocity`, assign an uniform value $U = U_\infty$ over all the patch.

Pressure : `zeroGradient`, it extrapolates values from the domain, thus it doesn't require any information.

Phase Fraction : `variableHeightFlowRate`, it takes the variable from the internal field and if it is inside the defined bounds it behaves like `zeroGradient`.

ATMOSPHERE This patch represents the behavior in the air outlet above the ship. It shouldn't see strong variations, but it must be well defined not to add instability to the solution.

Velocity : `pressureInletOutletVelocity`, it is a velocity condition for boundaries where pressure is specified; it gives a `zeroGradient` for outflow and the patch-normal of cell value for inflow (to find inflow velocity its own value) $U = 0$.

Pressure : `totalPressure`, it provides a condition on total pressure, by taking as input total pressure p_0, then inflow velocity will be subtracted to find pressure; in this case $p0 = 0$.

Phase Fraction : `inletOutlet`, if there is outflow it uses a fixed value, if there is inflow it applies a `zeroGradient` condition.

HULL It is the surface of the ship hence it cannot be penetrated by water and air and, since viscosity is considered, the velocity above it must be zero.

Velocity : `movingWallVelocity`, it applies a velocity condition in the presence of moving wall (necessary with `interDyMFoam`); in this case velocity is 0.

Pressure : `fixedFluxPressure`, adjust the pressure gradient to keep the boundary flux specified by the velocity condition. it is useful when body forces like gravity and surface tension are present.

Phase Fraction : `zeroGradient`, it extrapolates values from the domain, thus it doesn't require any information.

SYMMETRY `symmetry`, `side` and `bottom` boundaries have been defined as symmetry type then they have:

- zero normal velocity at the symmetry plane
- zero normal gradients of all variables at the symmetry plane

Now that the boundary conditions are defined it's time to discretize the domain to start resolving equations.

Chapter 9

Meshing

Meshing has been one of the most tricky and demanding tasks encountered during this research work. There are essentially three sources of difficulty.

GEOMETRY & PHYSICS Dealing with shape optimization, it becomes fundamental to preserve geometry curvatures and shapes. For this reason there are certain zones, especially underwater, that need to be "well-meshed". Thus fine discretization are required and, on 4-meters geometry, this means big grading.

Furthermore the presence of two phases requires interface between them (free surface) to be reasonably captured (effect of the bulb, as previously said, acts on hull wave system).

LOW NUMBER OF CELLS Optimization requires many data, this means many simulations. Hence the number of cells must be minimized to speed up calculations and optimization. A good trade-off between accuracy and computational speed have to be found.

OPENSOURCE MESHER Decision of using open-source tools can save money, but can provide some complications. For example, in this work the OpenFOAM internal mesher `snappyHexMesh` has been used. This has more then 70 parameters to control and have no user interface. Find the best setting can become very time-consuming.

Nevertheless, with an appropriate set-up, `snappyHexMesh` can give good quality hexahedral mesh.

In the next sections, after a brief explanation about the meshing tools used, two unstructured meshes are presented: a coarse mesh that has been called *coarseMesh* and a finer one that has been called *fineMesh*.

9.1 Meshing tools

blockMesh Taking as input the coordinates of the vertices `blockMesh` create the computational domain and then decompose it into a set of 3D hexahedral blocks. It is possible to define how many cells will be used to divide the domain, specify zones where more cells need to be used and the grading between these zones.

Furthermore `blockMesh` generates the patches where boundary condition will be imposed (see *Chapter 8*).

topoSet & refineMesh Even if `blockMesh` provides a first refinement, it is not enough to reach the required quality. Then two utilities are used: `topoSet` and `refineMesh`.

In few words, `refineMesh` take as inputs a set defined by `topoSet` and a direction along which it will split in two the cells.

surfaceFeatureExtract `surfaceFeatureExtract` extracts features from a surface or a file.

These will be used in `snappyHexMesh` to remove cells from the domain and to snap the remaining to the surface.

snappyHexMesh Given a triangulated surface (`.stl` or `.obj`), `snappyHexMesh` generates a three-dimensional mesh containing hexahedral and split-hexahedral cells.

Substantially, it takes an hexahedral background mesh (generated with *blockMesh*) and remove from it the cells inside (or outside) the triangulation; then it relaxes and adjusts the mesh to reach the required quality.

Since it represents the main stage of mesh generation, its steps are briefly explained.

- *Cell splitting at feature edges and surfaces* Cells intersected by the surface are split following the dictionaries specifications. Here a refinement can be specified by saying how many times cells will be split and how many cells will be used for transition to the rest of the mesh. Also surface refinement could be done to better maintain the geometry curvatures (there is a maximum angle over which surface can be more refined)

- *Cell removal* In `snappyHexMesh` dictionary there is a paramater called `locationInMesh` that specifies a point belonging to the mesh. If this point is inside the triangulated surface, all cells outside will be removed, conversely if it is placed outside (as in this case), cells inside the surface will be removed.

  As a rule, a cell is retained if 50% or more of its volume lies in the mesh.

- *Snapping to surfaces* This is the first iteration in `snappyHexMesh` procedure. Vertices near the surface are attracted to her, then mesh is relaxed. A control on mesh quality is made, if requirements are violated, the procedure is repeated until they're satisfied.

- *Adding layers* This is the most important step because it represents the spatial discretization in the boundary layer, so its quality directly influences y^+ and, consequently, the solution. The mesh is displaced normally away from the surface by a specified thickness. Again, the mesh is relaxed and a first quality control is made, if the result is bad the thickness is reduced and the procedure is repeated until satisfaction (if a satisfactory mesh isn't reached, layers won't be inserted). When quality is reached, layers are added and another check is made, if it fails, layers are removed and relaxation is solved again.

  It is important to highlight that layers addition is made on the existing mesh, and not to the surface; as a consequence, all the discretization steps before will influence the quality and the rate of convergence of `snappyHexMesh` procedure.

  All mesh quality requirements are specified in a dictionary called *meshQualityControls*.

9.2 Coarse & Fine Meshes

Spatial discretization is inspired to that of *DTCHull OpenFOAM validation case* [6].

In this work two meshes have been realized to allow benchmarking and validation of results.

As said in *Chapter 8*, to be sure that boundary conditions won't affect the results, a large domain is required, but, on the other hand, a fine discretization is necessary to catch two important phenomena: free surface and boundary layer. So refinements are necessary to give accuracy to the simulation, while maintaining its computational costs relatively low.

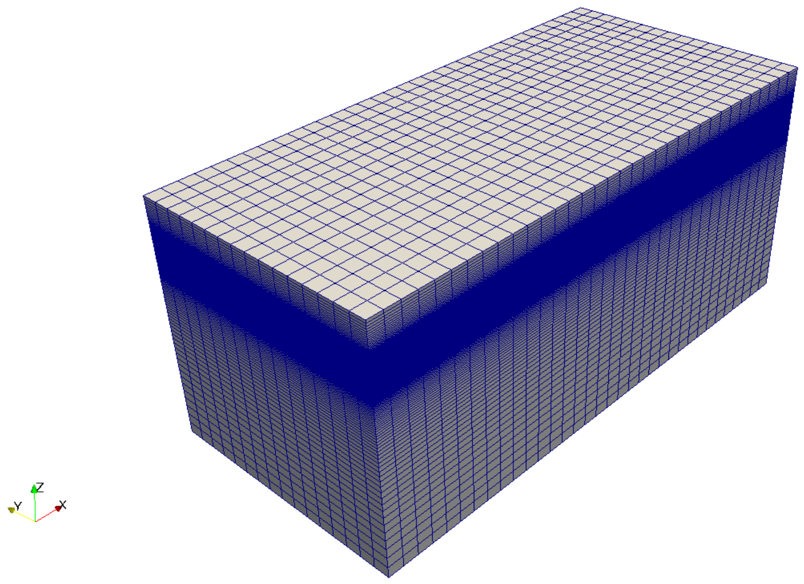


Figure 9.1: *Background mesh generated by* `blockMesh`

9.2.1 Background mesh

Both mesh generations start from the same background mesh generated with `blockMesh`. The approach is to divide the domain in cells with a base $(1 \times 1)[m]$ and variable height. In *Figure 9.1*, where a `blockMesh` output is shown, it is possible to see these cells.

9.2.2 Free Surface refinement

Even if free surface definition isn't a primary goal of this project, ensuring a good representation, especially near the hull, is necessary to make accurate resistance predictions. This is accomplished by defining different regions and grading in `blockMesh` dictionary. Is is important to note that for free surface resolution a refinement in the *z-direction* only is sufficient.

In the *fineMesh* a further refinement is made not to have too stretched cells near the hull and causes problem to `snappyHexMesh`. This will be done with the already mentioned tools `topoSet` and `refineMesh`.

In this case cells in the set defined by `topoSet` dictionary are split in two along the vertical direction z (as specified in one `refineMesh` dictionary).

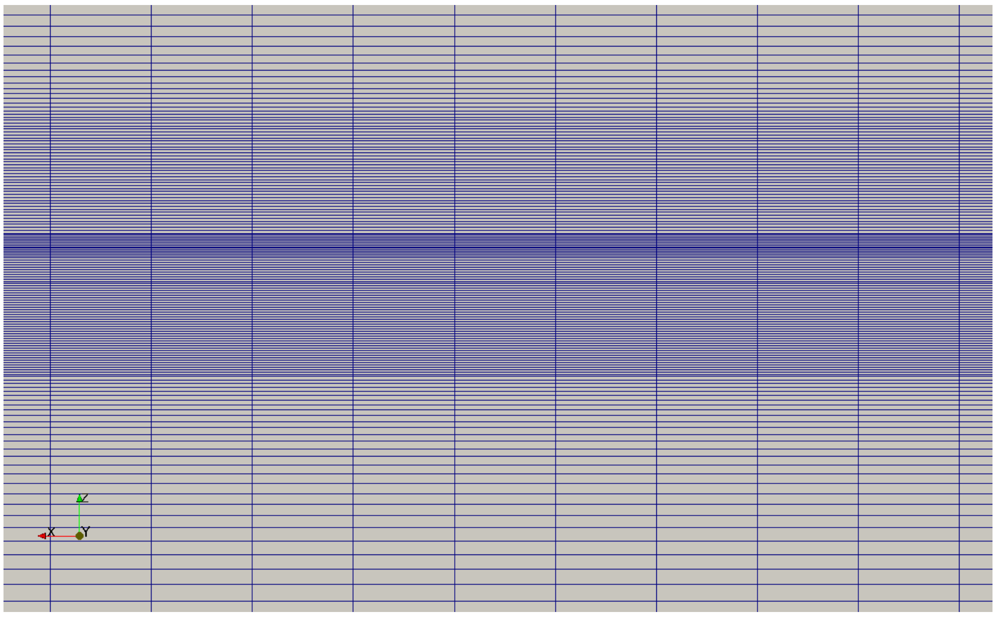


Figure 9.2: *Free surface refinement as specified in `blockMesh` dictionary*

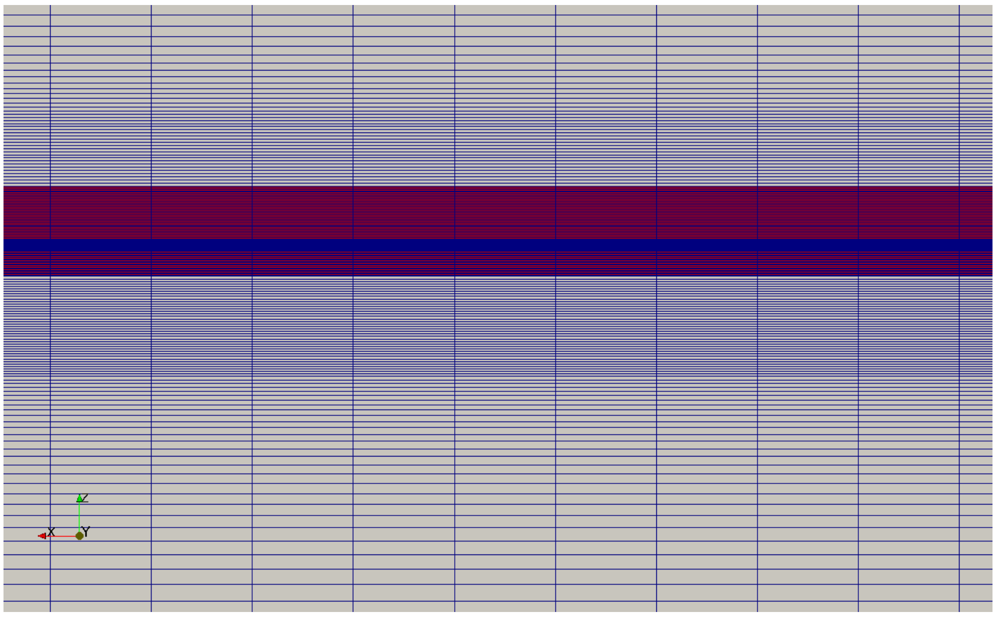


Figure 9.3: *Free surface refinement made with `topoSet` and `refineMesh`, zone in red corresponds to refined cells*

9.2.3 Near-Hull refinement

As said in the introduction to this section, a fine discretization is required near the hull, especially in the boundary layer.

The approach to this request consists in defining a series of refinement boxes nested inside each other, that contain hull geometry. Since refinement along the *z-direction* has been already made, in this context it is necessary to refine along x and y directions. So all the cells inside each box will be split in four littler cells (two along x and two along y). Given that these boxes are disposed like a "matrioska", in the inner box splitting will be performed a number of times equal to the number of boxes.

As shown in *Figure 9.4*, *fineMesh* has 7 boxes, while *coarseMesh* has 6. This means that cells in the inner box will have dimensions:

$$\textit{coarseMesh}\ \frac{1}{2^6} \simeq 0.016[m]$$

$$\textit{fineMesh}\ \frac{1}{2^7} \simeq 0.008[m]$$

To contain the hull geometry boxes are slightly bigger in the *fineMesh.*

Practically all these boxes are defined in different `topoSet` dictionaries and for each one `refineMesh` is applied. Note that in this case `refineMesh`, since has to split cells in x and y directions, will use a different dictionary from that used for free surface refinement (along z).

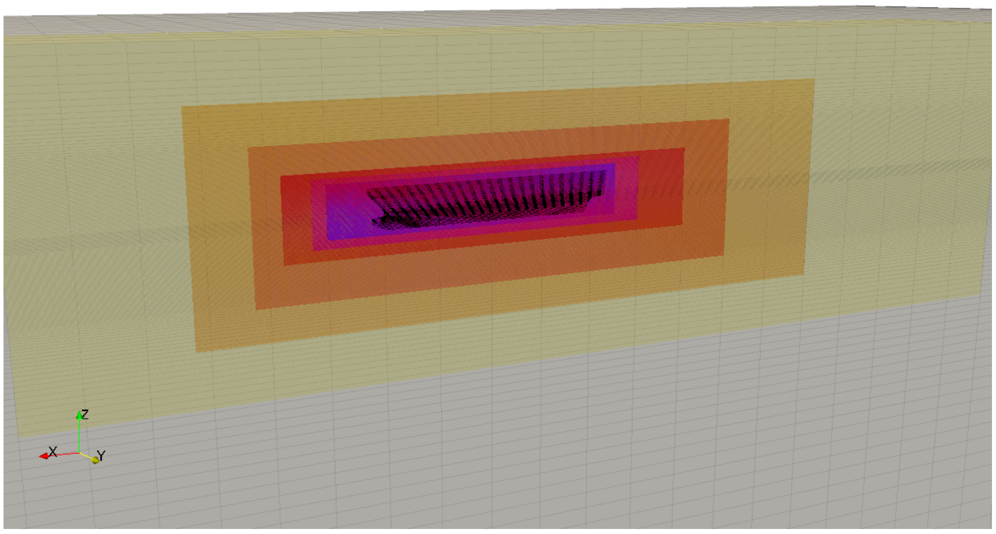
The presence of more than one boxes is necessary to ensure a slow grading in cell dimensions, ensuring numerical stability and accuracy. Having small and compact cells around the hull will help `snappyHexMesh` in reaching convergence and generating a good mesh.

9.2.4 Generating the Mesh

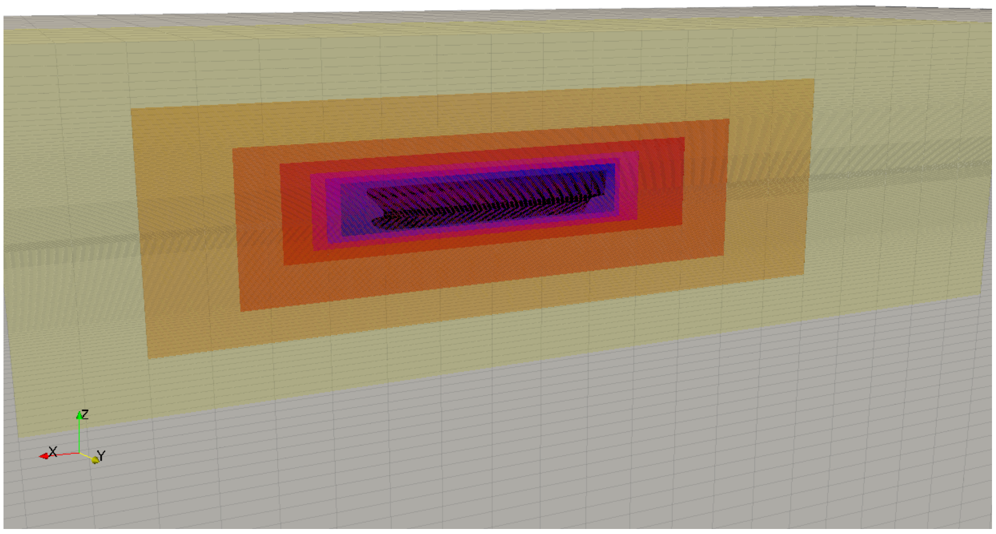
Once the refined background mesh are created `snappyHexMesh` is used to generate the two final meshes.

With regard to `snappyHexMesh` dictionaries, in both discretizations no surface or local refinement has been used, while in the adding layer procedure there is a difference in dictionaries set up.

In *coarseMesh* the number of layers to add has been set to 3, and the size of the furthest layer has been set to be 70% of the cells size near the hull and the minimum to be at least 25%.

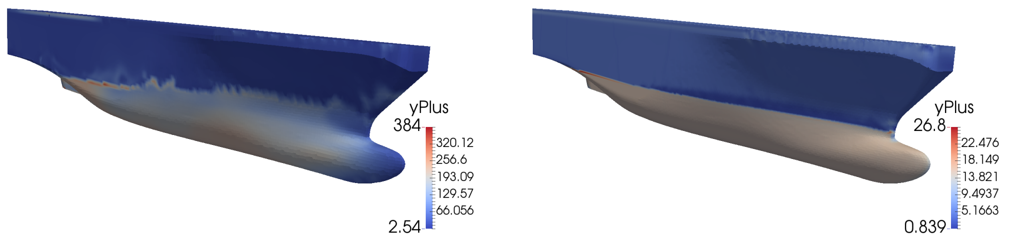


(a) coarseMesh



(b) coarseMesh

Figure 9.4: ***topoSet*** *boxes to refine near the hull geometry in the case of coarseMesh (a) and fineMesh (b)*



(a) coarseMesh (b) fineMesh

Figure 9.5: *Distribution of y^+ on the hull surface at $Fn = 0.331$ relative to the first iterations in the coarseMesh (a) and fineMesh (b) cases*

In *fineMesh* relative sizes value are the same, but to add three layers a different approach has been used. In fact, adding three layers simultaneously causes a bad covering of the hull. Then a good solution has been found by setting the number of layers to 1 and applying `snappyHexMesh` three times.

9.2.5 Checking quality

When talking about mesh quality, considerations are based more on users' experience than on a written theory.

The respect of the physics of the problem and the reasonableness of the results have been the main criteria to check the quality of the mesh. Since experimental results are available for the case study, these data have been used to check the numerical set up (see *Chapter 16*).

Even if experimental results could present errors, sometimes more than computations, having a similarity of results has given a measure about the quality of the work, helping in the set-up choice.

Since wall functions are used to solve for the turbulent flow, attention must be paid on the first layer thickness. Usually it is recommended to have a y^+ mean value between 30 and 500(see *Chapter 11*), however, since the $k - \omega$ *SST model* is used, it has been choose to extend this range from 1 to 600.

In this view, the mesh has been checked running some iterations and controlling the distribution of y^+ on the hull. These are shown in *Figure 9.5*.

Number of cells	$\sim 760,000$
Average y^+	~ 80

(a) *coarseMesh*

Number of cells	$\sim 2,800,000$
Average y^+	~ 7

(b) *fineMesh*

Table 9.1: *Features of the two spatial discretizations.*

9.3 Remarks

In shape morphing it is important that meshing dictionaries work well for all the cases; many geometries imply many meshes, so controlling the quality and adjusting the dictionaries every time can become tricky and time consuming.

Finally, considerations about speed and accuracy must be made to choose the one to employ in optimization. In *Chapter 16* a comparison with towing tank experimental results is presented. Also a comparison between the two meshes using an unsteady solver is presented.

Chapter 10

Temporal Discretization

As said in *Chapter 7*, investigations are made considering a *fixed trim condition* for the ship. So a steady state solution must be computed.

This 0-degree-of-freedom assumption is a common practice in ship resistance computation, since it allows to speed up a lot the calculations. Furthermore, in towing tank tests position of the hull seems not to vary significantly, making this simplification not too strong.

The big difference in calculation time is given by carrying out a pseudo-transient simulation with the use of the *Local Time Stepping (LTS)* technique.

For the purpose of this thesis, the choice of this method has been necessary, since it has represented a very good compromise between accuracy (comparing with experimental results) and computational costs. In the key of optimization this is an important requirement of the numerical model.

However, since *LTS* is not well documented in literature and its parameters are not so easy to manage, also a classical Euler implicit method with adjustable time-step has been tested for benchmarking (see *Chapter 16*).

10.1 Temporal discretization

When the aim of a simulation is to represent a transient behavior, for example the free surface evolution or the ship's trim and sinkage, an additional coordinate must be added. This is the *time coordinate* t.

This new coordinate, unlike the spatial ones, has no backward influence, i.e. something that happens at time t can affect the flow only in the future and not in the past. This *parabolic* nature allows to consider only few points of the temporal domain instead of fields as in the spatial case.

Before describing the time discretization methods it is important to introduce the concept of *Courant-Friedrich-Levy (CFL)* condition.

This criterion describes the stability of the solution, based on the fact that all initial information must be used to compute the forward time in order to have a *PDE*'s solution to converge. In other words, a small enough time-step must be used to allow that all information can propagate through the mesh. This can be written has:

$$CFL = \Delta t \sum_{i=1}^{N} \frac{u_i}{\Delta x_i} \leqslant CFL_{MAX} \tag{10.1}$$

where N are the dimensions of the problem, Δt and Δx_i the temporal and spatial length and u_i is the information. The CFL_{MAX} depends on the numerical schemes employed in the equation discretization.

Methods to discretize along time coordinate can be distinguished in *explicit* and *implicit*. As suggested by their names, the first give an explicit calculations of the value at a forward time $t + \Delta t$ since it depends only on quantity referred to time t, while the second gives an implicit solution, some quantity depends on $t + \Delta t$, then a system of equations must be solved.

Even if implicit methods have higher computational costs (need to solve a system of equation at each time step), they have the advantage to be unconditionally stable. This mean that larger time-steps can be used, leading to a faster convergence of the solution.

However, too large time-steps cannot be chosen; in fact it is necessary to ensure that the significant temporal scales are resolved in order not to hide some time-dependent features.

Thus, even if *explicit* methods require less calculation than *implicit*, the strong limitations of the *CFL condition* can lead to very small time step, augmenting a lot the computational costs. Furthermore the *Courant Number* is related to spatial discretization (equation (10.1)) and so, the finer the spatial discretization the finer must be the temporal one.

These limitations of *explicit* methods lead to the wide-spread use of *implicit* methods in the naval field.

10.2 Local Time Stepping

Usually when the temporal discretization is made with an implicit method a maximum *CFL* number is imposed for each cell. So time-step is adjusted each iteration to ensure that *Courant* value for the more critical cells. This allows to maintain a good accuracy of the solution.

In this case, as it has been seen in *Chapter 9*, there is a big difference in size between the cells near the hull and those far away ($\sim 2^7$). This implies that, if a maximum value of *Courant Number* is imposed, the smaller time-step in the critical cells will also be applied to the bigger ones, slowing a lot the simulation.

In this view it can be thought to apply a bigger time-step in the zone where the fields change "slowly" and the temporal scales are bigger. The *Local Time Stepping* approach is just this: applying a different and adequate time-step for each cell.

The time step is "maximized" in each cell according to a fixed maximum *Courant Number* (`maxCo`). This lead to a discontinuous time-step field that is smoothed in order to avoid errors from sudden changes.

It must be underlined that this procedure clearly violates the physics during all the "transient", for this reason time-steps assume the role of iterations. This non-physical aspect makes this procedure difficult to manage. Furthermore, the free surface evolution will be "killed" in the zone away from the hull.

Local Time Stepping has shown to be very powerful, leading to results very close to the experimental ones with a computationally cheap procedure. As for spatial discretization a comparison with a different scheme (classical Euler implicit) has been made (see *Chapter 16*).

Chapter 11

Turbulence modeling

In CFD the simulation of turbulence is one of the most important aspect to achieve reasonable and accurate results. Unfortunately it is also one of the most tricky.

The set up of turbulence modeling has been made following well known validation cases [6] and common guidelines [8],[7].

As stated by the 27th *International Towing Tank Conference*, the *Reynolds Average Navier-Stokes Equations (RANSE)* are the main workhorse for resistance computations, offering good accuracy, reliability and fast solution turnaround time [8].

For this reason Reynolds-Averaging approach is briefly explained in this chapter, with a brief description of the method adopted in this work, i.e. the $k-\omega$ *Shear Stress Transport* model.

11.1 Reynolds Averaged Navier-Stokes Equations

Turbulent flows are characterized by the presence of eddy structures of very different length scale. Accurately computing each of these scales would require about $10^9 - 10^{12}$ cells, making the calculation unaffordable. Thus, a variety of procedures have been developed to simulate the effect of turbulence, avoiding to calculate all the details of the turbulent flow.

In general the choice is between filtering and averaging. Since the first is too expensive for the purpose of this study, the averaging procedure has been chosen.

The main principle behind the so-called *Reynolds-averaging* is to decompose the flow variables in a *time-independent* ϕ and a *fluctuating* ϕ' component, to substitute them in the equations and then to average them over time.

The time-averaging properties lead to the following expression for the incompressible *RANS* equations:

$$\begin{cases} \boldsymbol{\nabla} \cdot (\rho \overline{\boldsymbol{v}}) = 0 \\ \\ \dfrac{\partial \rho \overline{\boldsymbol{v}}}{\partial t} + \boldsymbol{\nabla} \cdot \{\rho \overline{\boldsymbol{v}} \overline{\boldsymbol{v}}\} = -\nabla \overline{p} + \nabla \cdot \left(\overline{\boldsymbol{\tau}} - \rho \overline{\boldsymbol{v'} \boldsymbol{v'}} \right) + \rho \boldsymbol{g} \end{cases} \tag{11.1}$$

Comparing theses equation with those written in *Chapter 7*, one can note the appearance of anew term on the right-hand-side. This term is called *Reynolds Stresses Tensor*. So what Reynolds-averaging does it to collect all the instantaneous fluctuations in the 6 new unknowns.

Here comes into play the *Bousinnesq Hypothesis* which makes an analogy with Newtonian fluids by assuming that the *Reynolds stresses* are a linear function of the *mean velocity gradients.*

$$-\rho \overline{\boldsymbol{v'} \boldsymbol{v'}} = \mu_T \left[\nabla \boldsymbol{v} + (\nabla \boldsymbol{v})^T \right] - \frac{2}{3} \rho k \boldsymbol{I} \tag{11.2}$$

This assumption reduces the number of unknown from 6 to 2: the *turbulent eddy viscosity* μ_T and the *turbulent kinetic energy* k.

For incompressible flows, the equations can be rearranged by defining a *turbulent pressure* p [14]:

$$p \leftarrow p + \frac{2}{3} \rho k \tag{11.3}$$

In this manner, the only unknown that remains to compute is the *turbulent eddy viscosity* μ_T. It is on the way this quantity is expressed or computed that lies a great variety of turbulence model.

In the following section are presented the principle and the capabilities of the *Shear Stress Transport (SST) k-ω model* that is the one employed in this work. However for a better understanding also the $k - \varepsilon$ and the $k - \omega$ models are briefly explained.

11.2 Shear Stress Transport $k - \omega$ Model

The *k-ω* family of linear eddy viscosity models seems to be by far the most widely used ones [8].

Before describing the *SST* $k - \omega$ *model* it is necessary to brielfy introduce *standard* $k - \varepsilon$ and $k - \omega$ models, since *SST* is a combination of these two approach.

Both methods belong to the *two-equations* family of turbulence models. This class involves the resolution of two additional partial differential equations in order to locally compute the *turbulent eddy viscosity* μ_T.

Let's have a look at one of the earliest of these methods: the *standard* $k - \varepsilon$ model developed by Jones and Launder.

$\boldsymbol{k - \varepsilon}$

Like other models based on *Bousinnesq Hypothesis*, the $k - \varepsilon$ is based on the following expression for *turbulent eddy viscosity* μ_T:

$$\mu_t = \rho C_\mu \frac{k^2}{\varepsilon} \tag{11.4}$$

where C_μ is a calibration constant, k is the *turbulent kinetic energy* and ε is the *rate of dissipation of turbulent kinetic energy per unit mass due to viscous stresses.* Solving the following transport equations for k and ε a local value of μ_T can be computed:

$$\frac{\partial}{\partial t}(\rho k) + \nabla \cdot (\rho \boldsymbol{v} k) = \nabla \cdot ((\mu + \frac{\mu_T}{\sigma_k})\nabla k) + S_k \tag{11.5}$$

$$\frac{\partial}{\partial t}(\rho \varepsilon) + \nabla \cdot (\rho \boldsymbol{v} \varepsilon) = \nabla \cdot ((\mu + \frac{\mu_T}{\sigma_\varepsilon})\nabla \varepsilon) + S_\varepsilon \tag{11.6}$$

It must be kept in mind that the construction of this model is based on two important assumptions:

- *fully turbulent flow*
- *negligible molecular viscosity effects*

that establish the limits of this approach:

- *validity only for high Reynolds*
- *inability to reach the wall*

To account for this lack the so-called *low Reynolds* $k - \varepsilon$ model have been developed. These use damping functions to damp the turbulent viscosity while getting close to the wall.

$\boldsymbol{k - \omega}$

In this model the equation for ε is substituted by an equation for ω, where ω is called *specific turbulent dissipation* and represents the rate at which the turbulent kinetic energy is converted into thermal energy per unit time and unit volume.

$$\omega = \frac{\varepsilon}{C_\mu k} \tag{11.7}$$

Then *turbulent eddy viscosity* is given by:

$$\mu_T = \rho \frac{k}{\omega} \tag{11.8}$$

The two additional equations are:

$$\frac{\partial}{\partial t}(\rho k) + \nabla \cdot (\rho \boldsymbol{v} k) = \nabla \cdot ((\mu + \frac{\mu_T}{\sigma_k})\nabla k) + S_k \tag{11.9}$$

$$\frac{\partial}{\partial t}(\rho \omega) + \nabla \cdot (\rho \boldsymbol{v} \omega) = \nabla \cdot ((\mu + \frac{\mu_T}{\sigma_\omega})\nabla \omega) + S_\omega \tag{11.10}$$

This new equation has three advantages [14]:

- *it is easier to integrate*;
- *it is integrable also in the sub-layer without using damping functions*;
- *it is capable to deal with weak adverse pressure gradient.*

As pointed out by its inventor, $k-\omega$ model is accurate for both free shear flows and wall-bounded (attached boundary layer and mildly separation) [26]. But, unfortunately, this model has a strong dependence on the free stream values.

Looking at capabilities and flaws of the two models, they seems to be "complementary". The $k-\varepsilon$, thanks to its insensitivity to free stream, predicts with more accuracy away from the wall, while the $k-\omega$ behaves better in the boundary layer and with weak adverse pressure gradient.

These considerations have led to the development of the *Baseline (BSL)* $k-\omega$ model, that uses a *blending function* to switch from the $k-\omega$ and a rearrangement of $k-\varepsilon$ in terms of ω.

The *Shear Stress Transport* $k-\omega$ model represents a further improvement to the *Baseline*, by limiting the shear stress in adverse pressure gradient. Menter, the developer of these two methods, writes [23]:

"*It (BSL) has a performance similar to the Wilcox model, but avoids that model's strong freestream sensitivity. The second model (SST) results from a modification to the definition of the eddy-viscosity in the BSL model, which accounts for the effect of the transport of the principal turbulent shear stress. The new model is called shear-stress transport-model and leads to major improvements in the prediction of adverse pressure gradient flows.*"

11.3 Wall functions

On every solid surface, due to the fluid viscosity, a boundary layer develops. This layer of fluid can be divided in three regions:

- *viscous sub-layer* ($0 < y^+ < 5$), where the effect of viscosity dominates;
- *buffer sub-layer* ($5 < y^+ < 30$), where viscous and inertial effects are equal;
- *inertial (log-law) sub-layer* ($30 < y^+ < 500$), where the effect of inertia dominates.

These three sub-layers can be identified by the value of y^+ that is the adimensionalized normal distance from the wall:

$$y^+ = \frac{d_\perp u_\tau}{\nu} \tag{11.11}$$

where $u_\tau = \sqrt{\tau_w/\rho}$ is the velocity scale.

This subdivision of the boundary layer is schematized in *Figure 11.1*.

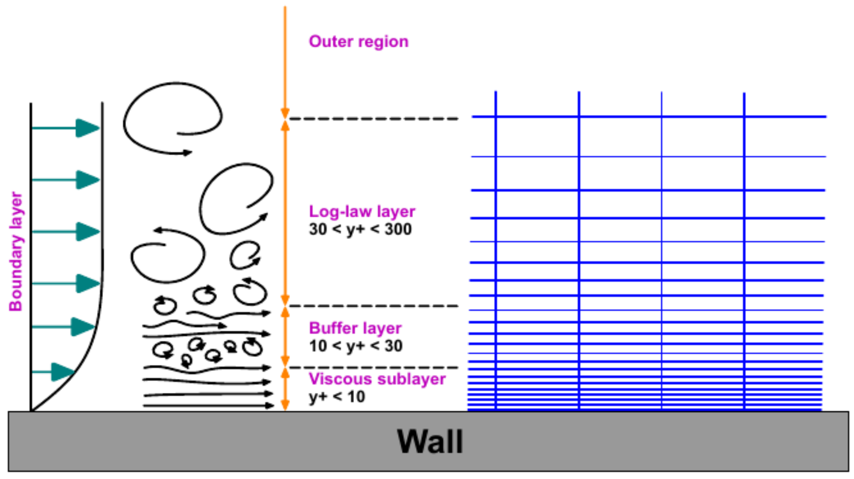


Figure 11.1: *Boundary layer subdivision and correspondent y^+ ranges. (courtesy of Wolf Dynamics srl)*

Turbulence models avoid the *buffer sub-layer*, because the high turbulent production, by placing the first cell center in the *viscous sub-layer* or in the *inertial sub-layer*.

The first option leads to accurate prediction of the boundary layer, but requires a very fine discretization near the wall, usually leading to unaffordable costs.

The second, combined by the definition an appropriate velocity profile between this point and the wall, significantly reduces computational costs while giving a good accuracy. This velocity profile is called *wall functionadn* its action is schematized in *Figure 11.2*.

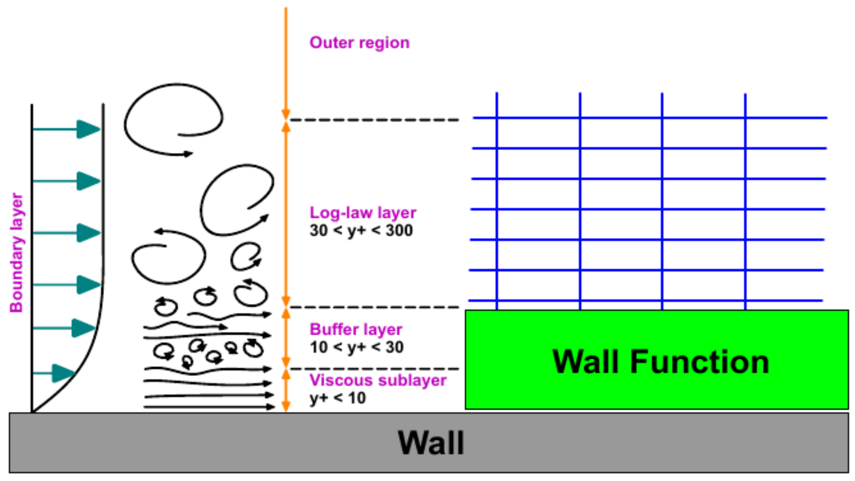


Figure 11.2: *Representation of the wall function approach. (courtesy of Wolf Dynamics srl)*

OpenFOAM offers several *wall functions*, among these the ones employed in this work are:

- *turbulent kinetic energy* k: `kqRWallFunctions`
- *specific dissipation rate* ω: `omegaWallFunction`
- *turbulent viscosity* ν_T: `nutkRoughWallFunction`

11.4 Remarks

The model chosen to simulate turbulence is the *Shear Stress Transport (SST)* $k-\omega$. This approach combines the good qualities of both *standard* $k-\varepsilon$ and $k-\omega$ models. Furthermore it accounts for the effect of Reynolds stress transport, allowing a better prediction in the presence of adverse pressure gradient.

The introduction of two new variables (k, ω) requires the definition of other boundary conditions. This is done with the use of *wall functions* that allow a coarser discretization near the wall by sumperimposing a theoretical velocity profile. Also a condition for the *turbulent kinematic viscosity* ν_T, that accounts for surface roughness effects, is imposed.

Usually for a correct use of wall functions the y^+ value must be in the range from 30 to 500, however the blending function of the *SST* allow to extend this range from 1 to 600.

This has been an important constraint in the mesh generation (see *Chapter 9*).

Part IV

Shape Morphing

Chapter 12

Shape Morphing Theory

In this thesis, in order to parameterize the shape of the bulb, the *Radial Basis Function* approach has been adopted.

This requires the definition of some *control points* that are the centers of some chosen function, the *Radial Basis Functions (RBF)*. These functions are used to generate a displacement field, by defining the value of the *RBF* in the control points and defining a support radius that defines their domain. In this manner the *displacement field* is defined as follows:

$$d(\boldsymbol{x}) = \sum_{i=1}^{n_c} d_i \psi(\|\boldsymbol{x} - \boldsymbol{c}_i\|) \tag{12.1}$$

where ψ is the *RBF*, x are the spatial coordinates, c_i are the control points coordinates and n_c the number of control points defined.

Calling $\boldsymbol{S}$ the points that define the undeformed geometry, the new, deformed geometry can be computed as:

$$\boldsymbol{S}^* = \boldsymbol{S} + d(\boldsymbol{x}) \tag{12.2}$$

When deforming only a part of the geometry, it becomes necessary to have an accurate control over deformable and undeformable portions of the surface and the possibility to prescribe the continuity condition between these regions [9].

In order to apply deformations only to a specific part Ω of the entire geometry, a *filter scalar function* w is defined so that:

$$\begin{cases} w = 0 \quad \text{outside } \Omega \\ 0 \leq w \leq 1 \quad \text{inside } \Omega \end{cases} \tag{12.3}$$

This filter function, in turn, depends on another scalar function $\phi(\boldsymbol{S}|\boldsymbol{\Gamma})$, called *geodesic level set function* or simply *distance function*, that represents the topological information about each point of $\boldsymbol{S}$ with respect to the boundary Γ.

Now it is possible to modulate the *RBF* displacement field with the filter function in order to obtain a constrained deformation of the geometry.

$$\boldsymbol{S}^{**} = \boldsymbol{S} + w(\phi(\boldsymbol{S}|\boldsymbol{\Gamma})d(\boldsymbol{x}) \tag{12.4}$$

The *distance function* chosen is the *geodesic distance function.* To evaluate this function, while guaranteeing a certain smoothness, a method based on the resolution of a time-dependent heat equation is used [9]. This method consists of the following steps:

- the heat equation $\frac{\partial u}{\partial t} = \Delta u$ (Δ is a *Laplace-Beltrami operator*) is solved in Ω over the time t_{end} and with $u = 1$ at Γ as boundary condition
- the vector field of the the normalized gradient is computed as $\boldsymbol{n} = \dfrac{\nabla u}{|\nabla u|}$
- the closest scalar potential ϕ is computed by solving a *Poisson* equation $\Delta\phi = -\nabla \cdot \boldsymbol{n}$ with a *Dirichlet* boundary condition along Γ

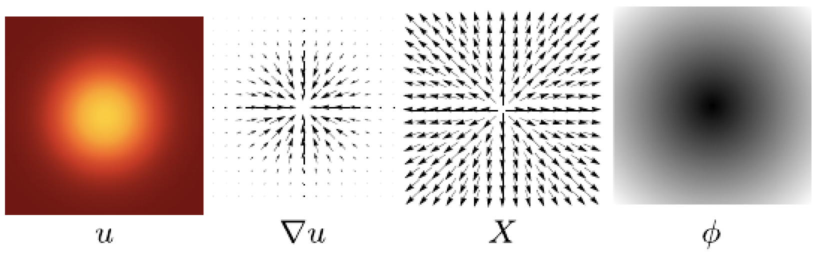


Figure 12.1: *Heat is allowed to diffuse for a certain period of time t_{end} reaching a u distribution, its gradient ∇u is computed and normalized $\boldsymbol{n}$ and then the function ϕ, whose gradient is $\boldsymbol{n}$, recovers the final distance (Crane et al. [19]).*

The evaluated function ϕ approximates the *geodesic distance*, approaching to the *true distance* as the time t_{end} goes to 0 [19]. In this case the desired level of smoothness is ensured by tuning t_{end}.

This has been just a short overview of the theory behind MiMMO/MIMIC shape morpher, for a deeper knowledge [9] and [19] are suggested. In the next chapter the set up of the shape morphing process will be described, showing how this theory is used in practice.

Chapter 13

Shape Morphing Set up

In this chapter the set up of the shape morpher is presented. This is done by creating an execution chain on a source file and then compiling it.

First of all, it is necessary to specify the control points, i.e. the RBF centers, and the displacements to apply in order to morph the geometry.

For example let's take the nose of the bulb as RBF control point and require that it must move 0.4 metres forward (*x-coord)* and 0.2 metres upward (*z-coord*). In

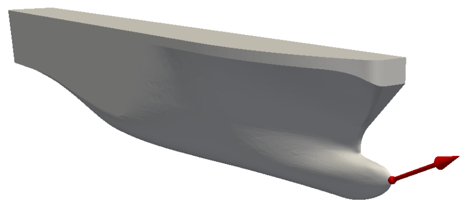


Figure 13.1: *Example of shape morpher inputs: control point (bulb's nose) and displacement ($x = 0.4[m]$ and $z = 0.2[m]$).*

the source code these input variables have been called `points` and `displ`.

MIMIC works by assembling an execution chain made with code blocks predefined by the user. First the blocks that will be used in this case are defined, then the execution chain is composed.

13.1 Definition of Code Blocks

This is the setup used in the optimization, thus the definition of blocks is made taking some precautions related to ship design.

`stlIn` this *MiMMOGeometry* object is created to read the input `.stl` geometry to morph (it also stores it in another *.stl*).

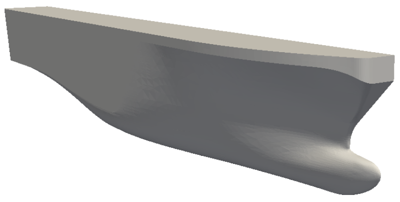


Figure 13.2: *Image of the input geometry*

`pcloud` this *ProjectCloud* object is created to project the control `points` on the target geometry. In this example (but also in the optimization process) the only one control point used is already on the surface, however this block is implemented to make the morpher more flexible.

`selbox` this *SelectionByBox* object is created to define a box that identifies the deformable portion of the whole geometry.

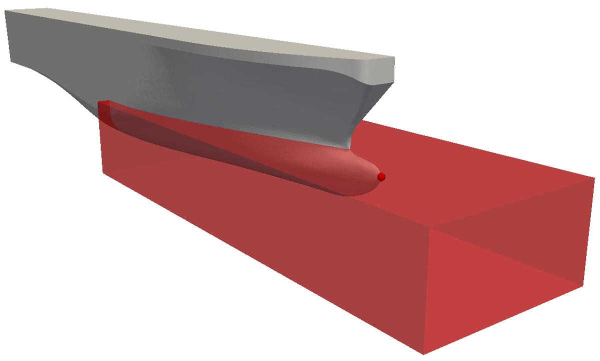


Figure 13.3: *Image of selection box that identifies the deformable part of the hull*

Looking at *Figure 13.3* it is possible to notice that this is defined in order not to change the main ship parameter, i.e. it arrives to amidships and to the forward point.

`sconstr` this *SurfaceConstraints* object is created to compute a filter function over the selected part of the geometry. This filter is function of the distance from the boundary between deformable and undeformable zones. The commercial code MIMIC allows to manage the continuity between the deformable and undeformable part (G^0, G^1, G^2) and the transition between them. However, for the shape morphing procedure developed in this thesis, these features haven't been fully exploited.

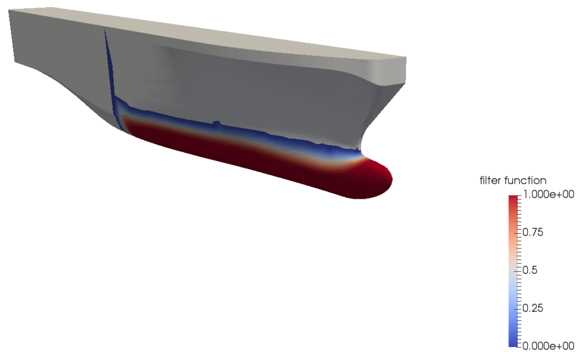


Figure 13.4: *Example of filter function computed on the geometry.*

`rcon` this *ReconstructScalar* object is created to reconstruct a scalar field over the whole input geometry.

`mrbf` this *MRBF* object is created to compute the deformation displacements of the nodes of the input geometry. This is done by defining the *support radius* of the *radial basis function* centered in the control `points`.

`applierS` this *Apply* object is created to modify the nodes of the input geometry using the displacement field that receives as input.

`stlOut` this *MiMMOGeometry* object is created to write the deformed geometry to an `.stl` file as output

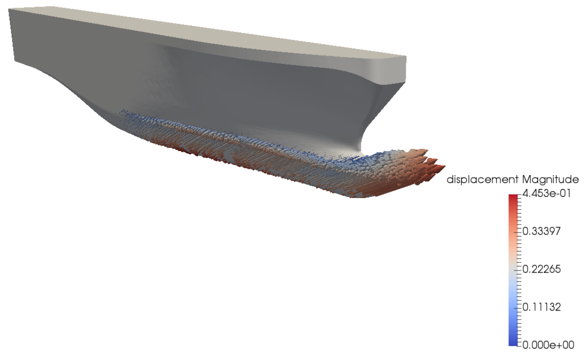


Figure 13.5: *Example of displacement field computed on the geometry.*

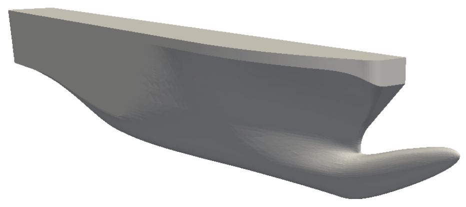


Figure 13.6: *Image of the deformed geometry as output*

13.2 Execution Chain

Once blocks has been defined it remains to define the connection between them and the data exchanged. In *Figure 13.7* there is a diagram of the execution chain.

Obviously the chain starts with the input of the geometry to be morphed; this is accomplished by the `stlIn` block. From this geometry is delivered to other five blocks: `selbox`, `pcloud`, `rcon`, `mrbf` and `applierS`. The `selbox` block takes the geometry and select, by intersection with the defined box, the deformable part of the geometry (geometry and boundaries). These data are sent to the `sconstr` block that, using the *HEAT algorithm* and the defined connection and continuity properties, compute the *filter function*. This scalar field is sent to `recon` that reconstruct the scalar field for the whole geometry.

The *reconstructed filter function* enter as input in the `mrbf` block that, taking the geometry from `stlIn`, the control points projected to the surface by `pcloud` and their displacements, compute the *distance function* field.

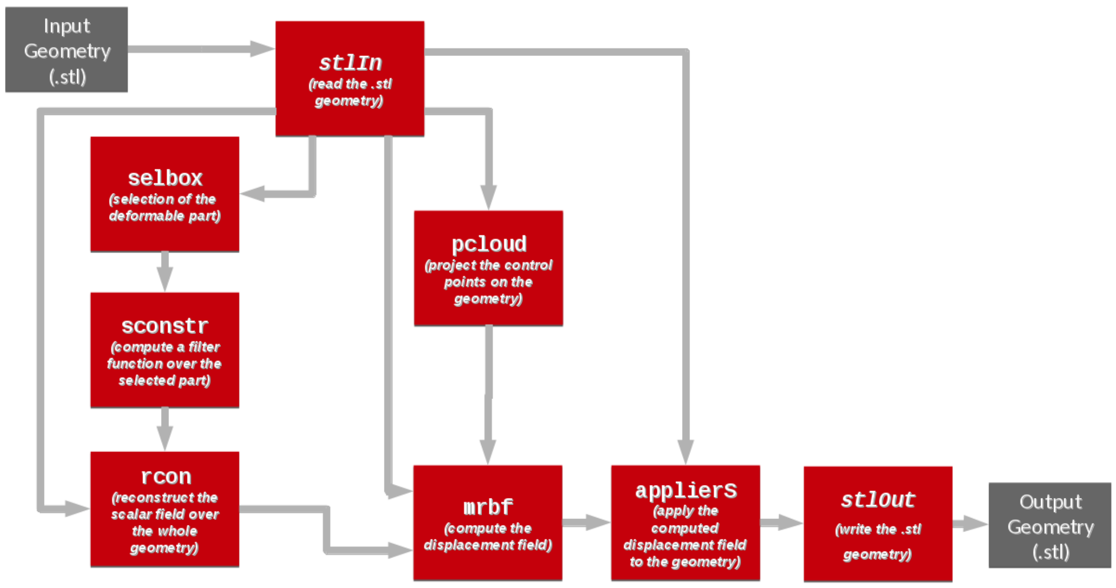

Input Geometry (.stl)
stlIn (read the .stl geometry)
selbox (selection of the deformable part)
sconstr (compute a filter function over the selected part)
rcon (reconstruct the scalar field over the whole geometry)
pcloud (project the control points on the geometry)
mrbf (compute the displacement field)
appliers (apply the computed displacement field to the geometry)
stlOut (write the .stl geometry)
Output Geometry (.stl)

Figure 13.7: *Workflow diagram of the execution chain used to morph the bulb.*

Finally, this *deformation field* is sent to the `applierS` that applies it to the undeformed geometry (from `stlIn`) and passes the result to the `stlOut` block that write the deformed `.stl` geometry file.

13.3 Input Parametrization

Once the source file is built it must be compiled to create the morpher. However for the purpose of optimization the input have to be parametrized.

During the optimization procedure, many geometries need to be created and the source has to be changed and the morpher to be compiled. To avoid this slowdown of the workflow, the morpher is slightly modified to take as input a file (`parameters.txt`) that contains the value of the two morphing parameters chosen: x and z displacements of the bulb's nose.

This step allows the interaction between MiMMO/MIMIC and Dakota, that update `parameters.txt` file.

In this chapter the theory behind the `MiMMO/MIMIC` shape morphing tool and the set up used to create the shape morpher have been presented. Together with the OpenFOAM set up explained in the previous part, they represent the two cornerstores of the optimization model: one morph the geometry, the other compute the resistance of the morphed geometry. In the next part, the linking element of the process, i.e. DAKOTA, will be presented.

Part V

Optimization

Chapter 14

Introduction to Optimization

Literally, optimization means *"an act, process, or methodology of making something (as a design, system, or decision) as fully perfect, functional, or effective as possible"*. This can be applied to many fields: finance, transports, design, health, energy systems, aerodynamic etc.

Each of these problems can be thought like a *black-box* that receives some inputs, let's call them *design variables*, and gives out some outputs, let's call them *quantity of interest* or *objective functions*.

So, given the problem, optimization means finding the set of design variables that minimizes, maximizes, equalizes or zeroes the quantities of interest.

In this chapter an overview of the various optimization methods is presented by referring to DAKOTA toolkit (*Chapter 4*.

14.1 Mathematical description

Mathematically speaking, optimization can be defined as:

$$\textit{finding} \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ x_n \end{bmatrix} \in \Omega \quad \textit{that} \quad \begin{matrix} \boldsymbol{minimizes} \\ or \\ \boldsymbol{maximizes} \\ or \\ \boldsymbol{equalizes} \end{matrix} \quad f_j(\boldsymbol{x}), \quad j = 1, ..., q \tag{14.1}$$

This is the more general statement, from which we can differentiate various kind of optimization problem.

The first aspect to highlight is the dimension n of the *design variable vector*. Obviously the higher it is the more expensive and complex will be the optimization study, and hence it becomes very important to know exactly which are the strictly necessary variables to characterize the problem.

The design vector $\boldsymbol{x}$ belongs to Ω that is a subset of $\mathbb{R}^n$ and is called *constraint set* or *feasible set*. Often this set is defined as $\Omega = \{\boldsymbol{x} \ : \ h(\boldsymbol{x}) = 0, \ \ g(\boldsymbol{x}) \leq 0\}$, where $h(\boldsymbol{x})$ and $g(\boldsymbol{x})$ are given functions called *functional constraints*, but there are cases where $\Omega = \mathbb{R}^n$ [13]. This is a first differentiation in optimization problems: the former is called *constrained optimization*, the latter *unconstrained optimization*.

Let's say, for example, that the scope is to maximize a certain objective function. Since its behaviour is unknown a priori, questions that come into mind are "How many maximum are there in the function?" " The scope is to find the global or all the local?" These questions highlight another fundamental difference: *local optimization* and *global optimization*. This point become extremely important when objective functions are *multimodal*, especially when using *Gradient Based Optimization*.

Another parameter to highlight is the number q of objective functions. In *single-objective optimization* $q = 1$ and then one or more optimal values are founded. In *multi-objective optimization* different outcomes need to be optimized simultaneously, so the result will be a set of optimal solutions, called *Pareto frontier*. This leads to make a trade-off analysis to find the better solutions.

There are other important feature of the problem, such as non-linearity and noise, that influence the choice of the optimization method to employ.

The great variety of optimization methods can be divided in two groups: *gradient based* and *derivative-free*, depending on the need of calculating the gradient or the hessian.

14.2 Classification of optimization methods

In Dakota User's Manual three criteria are used to differentiate all the existing optimization methods [12]:

- Problem Type is characterized by the constraints' type and by the linearity or nonlinearity of the objective and constraint functions. So:
  - *unconstrained problem* has no constraints;
  - *bound-constrained problem* has lower and upper bounds on the design variables;
  - *linearly-constrained problem* has linear and bound constraints;

  - *nonlinearly-constrained problem* has nonlinear, linear and bound constraints;

  Furthermore, based on the nature of all these constraints there could be:

  - *equality-constrained problem* if all are equality constraints;
  - *inequality-constrained problem* if all are inequality constraints.

  Particularly, if all the objective and constraint functions are linear there are called *Linear Programming (LP) problems.* If not, like it is common in engineering practice, they are called *NonLinear Programming (NLP) problems.*

- Search Goal of the optimization procedure can be to find the best feasible solution over the parameter space, i.e. *global optimization*, or can be to search the solution in a certain zone of the space, i.e. *local optimization.*

- Search Method represent the approach that is used to satisfy the search goal, i.e. the way a new design point with a more satisfactory objective function is located. This criteria differentiates in:

  - *gradient-based methods* that calculate the gradient of the response function to fin the direction of improvement;
  - *derivative-free methods* that avoid the computation of gradients (that often are expensive) by using many different approach (pattern search, genetic algorithms).

Often in engineering problems, especially in *Computational Fluid Dynamics*, the calculation of the quantity of interest requires lot of time and money. This can lead to the exclusion of many optimization methods, since they require to do a lot of iterations to find the optimum solution. An important way to deal with this problem is *surrogate modeling*, i.e. constructing a *low-fidelity*, but inexpensive, model that faithfully represents the behavior of a *high-fidelity* model by extracting information from a limited number of responses.

In this manner every type of optimization technique can be inexspensively applied at the surrogate level, that is inexpensive. This strategy is called *Surrogate-based Optimization (SBO)* and since it is the one used in this research work will be better explained in *Chapter 15.* Before doing so, an overview of the common available techniques is made and the choice is motivated.

14.3 How to choose an optimization method?

When facing an optimization problem there is a big assets of methods that can be used. However, depending on the case being studied, some will be faster and more accurate. So the best method have to be chosen.

This choice can be made by looking at the features of the problem to be optimized. In this view the main questions to ask are:

- *Does the black-box give smooth or noisy outocomes?*
- *Is the behavior unimodal or multimodal?*
- *How much does it cost obtaining the outcomes?*
- *Are interests focused on a single objective or more?*

Generally the *gradient-based* methods represent the most efficient solution, guaranteeing a good convergence rate [17]. These methods essentially looks for the optimal solution moving thorough the parameter space guided by the gradient information. This leads to three major problems:

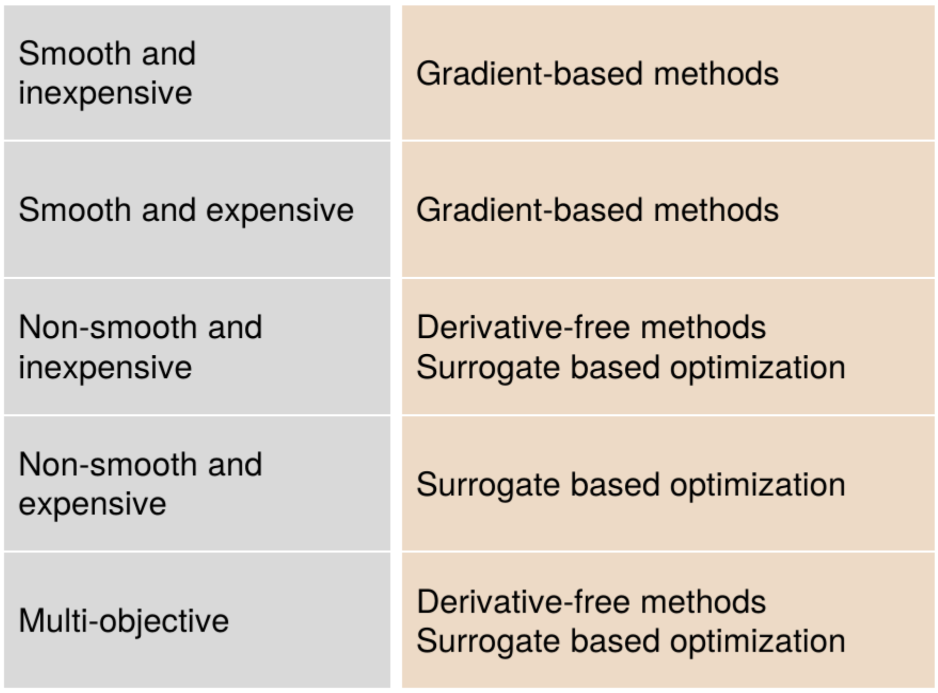
- knowledge of the gradient could become very heavy when the data acquisition is expensive;
- the optimum search must start from a design point (or more than one) and this must be chosen carefully because it will affect the convergence to the optimal solution and the time required;
- the presence of noise could "confuse" the search process, by locally corrupting the gradients.

A *gradient-based* optimization method does its best when the gradient computation is inexpensive and the problem behavior is smooth and single-modal or when the behavior is well known.

Derivative-free methods can be a good alternative when derivative computations are too expensive. They require only outcomes values and not their variation through the parameter space. This makes them very handy when globally optimizing a problem with a noisy and multimodal behavior. However, they have the fee of requiring a lot of outcomes to reach an accurate solution. So when the objective function evaluation is expensive these methods should be abandoned.

When the outcomes are expensive and the behavior is unknown a priori a good solution is *surrogate-based* optimization. With a limited number of function evaluations the behavior can be modeled and then analyzed. This allows to get information about the case study and, if a good surrogate has been constructed, each optimization method can be applied, since at the surrogate level they become inexpensive.

All these considerations are summarized in *Figure 14.1.*



Smooth and inexpensive	Gradient-based methods
Smooth and expensive	Gradient-based methods
Non-smooth and inexpensive	Derivative-free methods Surrogate based optimization
Non-smooth and expensive	Surrogate based optimization
Multi-objective	Derivative-free methods Surrogate based optimization

Figure 14.1: *General guidelines for the choice of the best optimization approach considering the features of the case being optimized. (Courtesy of Wolf Dynamics srl)*

Chapter 15

Surrogate Modeling

The shape optimization of the bulb presented here follows the *surrogate-based* approach. Essentially this means that a low-fidelity model is constructed to substitute the high-fidelity model (in this case the CFD simulation).

Before dealing with the various methods to build a surrogate, it is necessary to explain what a surrogate is and which advantages it takes.

In this instance, interest is focused on finding the optimal bulb geometry, i.e. the one that minimizes hydrodynamic resistance. For example, if only one design variable is considered, the behavior of the ship could be represented by a curve. Thus the optimization will mean to find the "lower" point of this curve.

The problem is that to reproduce this curve point by point would require a lot of simulations and each of these would involve hours of computation. Furthermore it is important to remember that the resistance behavior is unknown a priori and, except some physical intuitions, practically unpredictable. For this reasons it is clear that using gradient-based methods or genetic algorithms could become heavily expensive, especially if the starting point is badly defined. Here advantages of surrogate modeling comes into play.

In fact, when the behavior is unknown, a first thing to do is exploring the variable design space, making some observations. This step gives some insights of the problem. For example, the presence of minimums and maximums can be guessed and with this basic information gradient-based and genetic algorithms can be used more easily.

Otherwise the entire behavior can be reconstructed, passing from some discrete points to a continuous function called *surrogate model* (also known as *meta-model* or *response surface*). This means that the minimum research or any other analyses, won't be made using an expensive high-fidelity model, but on a surrogate of it.

In other words, the high-fidelity model (i.e. CFD simulation) will be used only to calculate a certain number of outcomes. Then these values will be used to build a meta-model (or surrogate) that predicts responses in other design points.

So, once a surrogate model is built, evaluating the quantities of interest becomes inexpensive, and any kind of optimization, calibration and investigation can be made.

In this view, surrogate based optimization has been an almost forced choice for the case being studied here.

The following section deal with the key stages of surrogate modeling procedure. Subsequently, the two approach that are employed in this work are described and the set-up of the optimization procedure is defined.

15.1 Key stages

15.1.1 Screening

First of all, the minimum number n of design variables that can highlight the black box behavior have to be defined.

$$\boldsymbol{x} = \{x_1, x_2, ..., x_n\} \tag{15.1}$$

Then k of this *n-vectors* have to be recruited to create the design space in which optimization will be performed.

$$\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_k\} \tag{15.2}$$

In this process many techniques can be used to choose the adequate combinations of design variables. Example of these are *full factorial sampling* or *Latin Hypercube sampling.*

15.1.2 Observing

Once the design space is defined, observations can be performed for each of the k chosen configurations. These could be done by experimental measures, data collections or, as in this case, code simulations.

This step can be see like the act of passing the *design vectors* as input to a black-box that gives back as output some *objective functions* to be monitored and optimized.

If, for example, only one objective function is considered, the results of this phase will be a set of k data pairs:

$$\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_k, y_k)\} \tag{15.3}$$

These represent points in the parameter space, called *training points*, and their collection is called *training data-set*.

15.1.3 Learning

This step is the essence of surrogate modeling. The goal is to learn from the obtained data pairs, that is searching through all the possible functions the one that better fits the training data-set. The bad news is that the number of candidates function is infinite, the good one is that the overwhelming majority of these are unreasonable. On the manner such bad candidates are filtered out lies the variety of surrogate modeling approaches.

The one adopted here is to choose a structure for the surrogate model $\widehat{f}$, defined by *structural* and *tuning parameters*. The structural ones are computed to fit the training data-set and then the best tuning parameters are searched to find the most "likely" surrogate with that structure.

15.1.4 Facing the noise

Usually the term *"noise"* is referred to random errors that corrupts physical measurements. Then it is related to something that it's not repeatable.

Computational experiments are *deterministic*, hence what is called *"noise"* is caused by model inaccuracy, that is systematic. Keeping in mind this is especially important when using Gaussian process, in which deterministic results are viewed as realizations of a stochastic process [11].

So, during the tuning of the surrogate, considering the noise is very important in order not to fit the data at too fine a scale. This problem is called *overfitting* and has to be avoided.

15.2 Polynomial & Kriging

As it will presented in *Chapter 17*, in this work the first approach to bulb optimization has been made considering only one design variable. To better understand the building process, for the one-design-variable case the surrogate has been constructed using an in-house Python script.

The theory used to develop this code is presented in the following to give the reader a brief introduction to the mathematics behind surrogate modeling.

15.2.1 Least-squares Polynomial Regression

Equation (15.4) shows the structure of a polynomial response curve, where the polynomial coefficients $\boldsymbol{w}$ are the *structural* parameters and the polynomial degree m is the *tuning* parameter.

$$\widehat{f}(x, m, \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2 + ... + w_m x^m = \sum_{i=0}^{m} w_i x^i \tag{15.4}$$

Now, supposing that the polynomial degree is known, let's find the polynomial coefficients $\boldsymbol{w}$.

Applying a polynomial equation to each training point a linear system can be constructed as follows:

$$\begin{bmatrix} 1 & x_1 & x_1^2 & ... & x_1^m \\ 1 & x_2 & x_2^2 & ... & x_1^m \\ ... & ... & ... & ... & ... \\ 1 & x_n & x_n^2 & ... & x_n^m \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ ... \\ w_m \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ ... \\ r_n \end{bmatrix} \tag{15.5}$$

Writing in a more concise form:

$$\boldsymbol{\Phi} \cdot \boldsymbol{w} = \boldsymbol{r} \tag{15.6}$$

where $\boldsymbol{\Phi}$ is the so-called *Vandermonde matrix* and $\boldsymbol{r}$ is the response vector.

Generally, to solve this system a *least-squares analysis* is employed. This means finding the vector $\boldsymbol{w}^*$ that minimizes the residuals $||\boldsymbol{\Phi}\ \boldsymbol{w} - \boldsymbol{r}||$. Using least-squares approach, the more "likely" estimation of $\boldsymbol{w}$ can be calculated as:

$$\boldsymbol{w} = (\boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \boldsymbol{r} = \boldsymbol{\Phi}^+ \boldsymbol{r} \tag{15.7}$$

Where $\boldsymbol{\Phi}^+$ is called *Moore-Penrose pseudo-inverse* of $\boldsymbol{\Phi}$.

At this point, once the coefficients are determined, the polynomial surrogate of the chosen order can be plotted. Unfortunately, in building surrogates, order of the polynomial is usually unknown a priori.

In this view, one could point out that the higher polynomials fits well the data and this is true. But in surrogate modeling danger of *overfitting* the noise must be kept in mind.

Then another method must be used to *tune* the polynomial degree m to better fit the data. Here *Leave-One-Out-Cross-Validation* is used.

First of all, it is necessary to have a measure of the model accuracy. This can be done by defining a *loss function* representing the difference between model and observations. Obviously to do this other design points, out of the training data set, need to be used; these are called *test data-set*.

As previously said, observations could be very expensive and time-consuming and to have a set of data used only for testing may not be convenient.

A way to overcome this limit is to split the available data set into q subsets and build the model without considering one of these; this will be used later as test data-set.

Rotating the left-out subset, q loss functions can be computed and that can help finding the best parameters.

This procedure is called *"q-Fold Cross Validation"*, when $q = n$ it is called *"Leave-One-Out Cross Validation"*.

15.2.2 Kriging Interpolation

The structure of the surrogate built with Kriging is given as the sum of a *trend function* $t(x)$ and a *gaussian process error model* $\varepsilon(x)$.

$$\widehat{f}(x) = t(x) + \varepsilon(x) \tag{15.8}$$

As described by *"Dakota v6.5 Theory Manual"* [12], Kriging interpolation involves mainly three steps:

- *choice of a trend function*
- *choice of a correlation (kernel) function*
- *estimation of correlation parameters*

The trend function could be a known costant (*simple kriging*), a general polynomial obtained with least-squares regression (*universal kriging*) or an unknown constant value (*ordinary kriging*).

As in many other Gaussian processes, a Bayesian approach is used, in the sense that the observed responses are viewed as if they come from a stochastic process [11]. Under this assumption, each response will have its own *expected value* and *covariance function*:

$$E\left[r(x_i)\right] = \boldsymbol{f}^T(x_i) \cdot \boldsymbol{\beta} \tag{15.9}$$

$$Cov\left[r(x_i), r(x_*)\right] = \lambda \cdot kernel(x_i, x_*|\theta) \tag{15.10}$$

where $\boldsymbol{f}^T(x)$ are trend function basis and $\boldsymbol{\beta}$ their weights, x^* is a design point outside the training data-set, λ is the process variance and θ is the correlation function parameter.

Collecting all responses, an *observed "random" vector* is defined:

$$\boldsymbol{r} = \{r(x_1),\ \ldots\ , r(x_n)\} \tag{15.11}$$

Then, by definition, the joint distribution of $\boldsymbol{r}$ satisfies [22]:

$$\boldsymbol{r} \sim \mathcal{N}_n\left(t(x), \lambda \boldsymbol{K}\right) \tag{15.12}$$

where $\boldsymbol{K}$ is the $n \times n$ matrix of correlations between training points (kernel function applied between all training points).

Assuming to know the trend and kernel functions parameters it is possible to compute *conditional expected value* and *conditional variance* of the process at an *untested location* x^*:

$$E\left[r(x_*)|r(x)\right] = \boldsymbol{f}^T(x_*)\boldsymbol{\beta} + \boldsymbol{k}_*^T \boldsymbol{K}^{-1}(\boldsymbol{r} - \boldsymbol{F}\,\boldsymbol{\beta}) \tag{15.13}$$

$$Var\left[r(x_*)|\boldsymbol{r}\right] = \lambda\left(1 - \boldsymbol{k}_*^T \underline{\boldsymbol{K}}^{-1}\boldsymbol{k}_*\right) \tag{15.14}$$

where $\boldsymbol{k}_*$ is the vector of correlations between the untested point and the training points, and $\boldsymbol{F}$ is the $n \times q$ matrix of all q trend basis functions at training points.

In the Python script the *squared exponential* is chosen as *correlation function*:

$$kernel(x, x^*|\theta) = e^{-\theta(x-x^*)^2} \tag{15.15}$$

As can be seen by its structure, this function correlates the outputs depending on the distance between inputs: very close inputs are strong correlated (~ 1), while distant inputs are weak correlated (~ 0).

This correlation can be tuned by varying the parameter θ: augmenting its value will enhance correlation, zeroing it will make outputs independent between each other. Here θ is a scalar because only one design variable is considered; if for example also morphing in z direction is considered, two different θ can be defined.

Another parameter that could be used for tuning is the exponent; it could be varied from 0 to 2 [22], but not to complicate the implementation this step has been avoided.

At this stage a criterion to tune the surrogate have to be defined; the one chosen in this context is *Maximum Likelihood Estimation.*

Essentially, tuning the kriging means to find the most "likely" multivariate normal distribution that fits the training data. Having parametrized mean and covariance functions (equations (15.9),(15.10)), optimization is reduced to their tuning parameters $\boldsymbol{\beta}$ and θ [22].

Fixed a polynomial degree, a polynomial trend function can be found using a least-squares analysis as in the previous section. Also in this case *Leave-One-Out-Cross-Validation* could have been used to find the order.

Now only θ remains to be computed.

To discriminate which is the optimal value, a measure of how "likely" is the surrogate need to be introduced. This can be accomplished by the *likelihood function.* Considering the logarithm form:

$$\log f(\boldsymbol{R} \mid \boldsymbol{x}, \lambda, \theta) = -\frac{n}{2}\log 2\pi - \frac{1}{2}\log(\lambda^n|\boldsymbol{K}| - \frac{1}{2\lambda}(\boldsymbol{R} - \boldsymbol{F}\,\underline{\beta})^T\underline{\boldsymbol{K}}^{-1}(\boldsymbol{R} - \boldsymbol{F}\,\boldsymbol{\beta}) \tag{15.16}$$

taking the negative and neglecting terms that don't influence optimization:

$$NLL \quad \propto \quad m\log\lambda + \log|\boldsymbol{K}| + \frac{1}{\lambda}(\boldsymbol{R} - \boldsymbol{F}\,\boldsymbol{\beta})^T\boldsymbol{K}^{-1}(\boldsymbol{R} - \boldsymbol{F}\,\boldsymbol{\beta}) \tag{15.17}$$

Thank to the properties of *Negative Log Likelihood (NLL)* gradients, it is possible to calculate an optimum λ for each iteration over θ . Hence the problem reduces to find θ that minimizes *NLL* [22].

The two methods described here are just a scrap of the surrogate modeling world. However they present very interesting features.

The big difference between them is that the first (polynomial) is a regression, so it uses the information given by the training data-set to predict a general behavior, while the second(kriging) is an interpolation, hence it includes the training points in the surrogate and predicts others from these.

This concept is extremely important in the interpretation of results. In fact, the "freedom" of polynomial regression with respect to the training points, will lead to a smooth surrogate. The Kriging instead will pass through all the training points, with the risk of overfitting the noise, but with the possibility of catching some particular physical trend. A good compromise between this could be the use of a ”nugget”. This option add a term to the diagonal of the correlation matrix, accounting both for measurement errors and alleviating ill-conditioning of the matrix [12].

Part VI

Results

Chapter 16

Numerical Model Benchmarking

During the optimization process, the outcomes from a *black-box* are compared while varying input design variables. In this way it is possible to find the best configuration for the case study. Obviously the optimal configuration mainly depends on the quality of this *black-box*. Inaccurate objective calculation will lead to the wrong optimum.

In this instance the *black-box* corresponds to the *OpenFOAM* case described in the *Part III*.

The accurate set-up of this numerical model has been made taking experimental results as reference. These data come from towing tank resistance measurements realized by *Schiffbau-Versuchsanstalt Potsdam GmbH* for the *Fincantieri*'s hull subject of this study.

It is important to highlight that also experimental results could present some errors (sometimes bigger than numerical), hence they could represent a good reference, but not a true index of accuracy. In this view, the case set up hasn't been influenced by these experimental results, but only compared.

Furthermore, since the focus is on the optimization procedure rather than on achieving a very accurate solution, interest is directed more toward trends rather than values.

In this chapter two benchmarks are presented to show the reasonableness of the numerical model. The first one is a comparison between numerical and experimental velocity-resistance trends. The second one is a comparison between the results given by four different numerical models:

- *Local Time Stepping* and *coarseMesh*;
- *Local Time Stepping* and *fineMesh*;

- *Global Time Stepping* and *coarseMesh*;
- *Global Time Stepping* and *fineMesh*;

16.1 Catching the Resistance-Velocity Trend

The experimental results available refer to a typical range of ship's sailing speeds: from 16 to 21 knots.

Using the *Froude similarity* described in chapter 5, the model-scale velocities have been calculated and implemented in *OpenFOAM* dictionaries:

$$V_{model} = \frac{V_{ship}}{\sqrt{\frac{L_{ship}}{L_{model}}}} \tag{16.1}$$

where the scaling factor $\lambda = \frac{L_{ship}}{L_{model}}$ is equal to 20 and the gravitational acceler-

Full-scale [knots]	16	17	18	19	20	21
Model-scale [m/s]	1.840	1.955	2.071	2.186	2.301	2.416
Froude Number *Fn*	0.294	0.312	0.331	0.349	0.367	0.386

Table 16.1: *Model-scale velocities according to Froude similarity*

ation is $9.81[\frac{m}{s^2}]$. For each one of these velocities a solution is computed with the OpenFOAM model.

To monitor the convergence of *LTSinterFoam* to the steady-state solution, resistance is plotted against "time-step" (since a Local Time Stepping technique is used the term iteration is more appropriate). Outputs of the six simulations are shown in *Figure 16.1.*

In this context the total resistance values are taken by averaging the solution over the last 4000 iterations. This is done to filter the influence of numerical noise, especially at high velocities. These values are collected in *Table 16.2* with the percentage error from the experimental values.

These data are reported in *Figure 16.2*, where experimental and numerical trends are compared.

As expected resistance increase with velocity, furthermore the OpenFOAM values are close to the measured ones. It can be noticed that, for low velocities, these

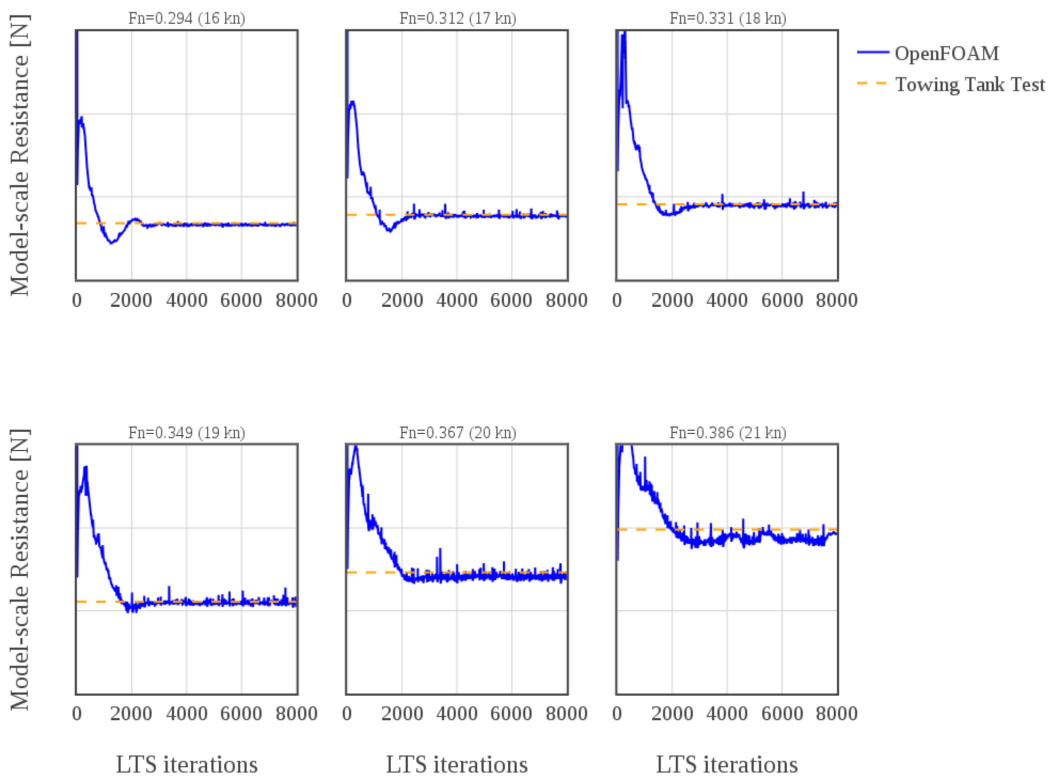


Figure 16.1: *Convergence to steady-state solution for the six Fn considered. (Numerical values are not shown because data is confidential)*

differences are small, confirming the reasonableness of the model. For high velocities some neglected effects, like ship's trim, breaking wave, turbulent structures and separation, become important and could lead to the diverging of the two trends.

However these results have been considered physically acceptable for the purpose of this study.

16.2 Numerical dependencies

The resistance-velocity trend is reasonably caught by the OpenFOAM numerical model. This is enough for the purpose of the optimization study investigated here. Especially because the simulation has demonstrated to be very light from the computational point of view, reaching the solution in a fast manner. This lightness has been one of the main keys in setting up the case.

The features that speeds up a lot the calculation are two: a mesh with a low number of cells (*coarseMesh*) and the use of the *Local Time Stepping* technique

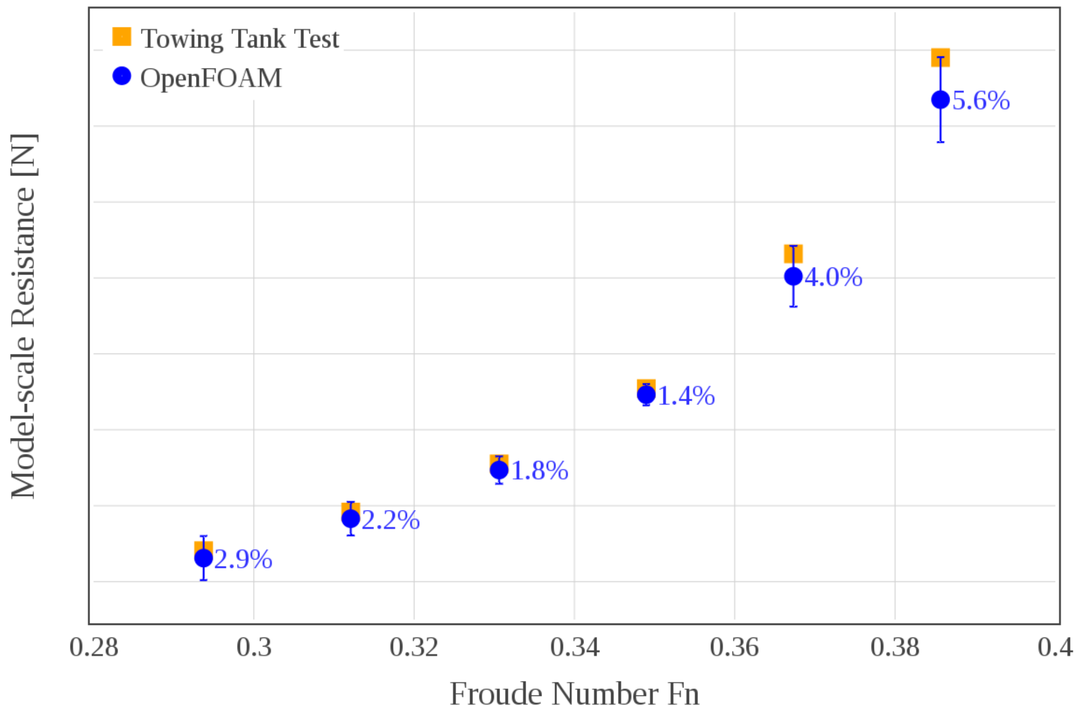


Figure 16.2: *Comparison between experimental and numerical resistance trends with increasing velocity. (Numerical values are not shown because data is confidential)*

to reach the steady solution. So it is worthwhile to perform a further validation by comparison to an ”heavier” numerical model, i.e. a finer mesh (*fineMesh*) and `interFoam` used as unsteady solver. For details about the two meshes a look at *Chapter 9* is suggested. Regarding the ”finer” temporal discretization, an Euler implicit scheme with adjustable time-step has been used. Let’s called this set up *Global Time Stepping (GTS).*

In *Figure 16.3* a comparison between the convergence of the total resistance computation in the four numerical set up is presented.

Despite the different types of iteration compared, this result leads to some important considerations. Firstly, all the models converge to a value consistent with the experimental. Secondly, the refinement of the spatial discretization ”moves” the unsteady result toward the towing tank test value.

The computational time required to reach a stable solution is very different, underlining the effect of a pseudo-transient analysis in speeding up the simulation. In simple terms, the green curve involves ”hours”, the blue one ”days”.

Again it must be underlined that the purpose of this thesis is not a validation of

Full/model-scale Velocity [kn] / [m/s]	**Froude Number Fn**	**OpenFOAM Resistance* [N]**	**Towing Tank Test Resistance* [N]**	**Error %**
16 / 1.840	0.294	-	-	2.9
17 / 1.955	0.312	-	-	2.2
18 / 2.071	0.331	-	-	1.8
19 / 2.186	0.349	-	-	1.4
20 / 2.186	0.367	-	-	4.0
21 / 2.416	0.386	-	-	5.6

Table 16.2: *Numerical and experimental results for different Froude Number (*Resistance data are under confidentiality agreement with Fincantieri)*

local time stepping technique, nor the investigations of the better numerical model for the ship's resistance study. These subjects go beyond the analysis done here.

Rather, the aim of the comparison presented in this chapter has been to validate the reasonableness of the results given by the numerical model. A better understanding of the hydrodynamics of the ship and improvements on the dictionaries are open for further developments.

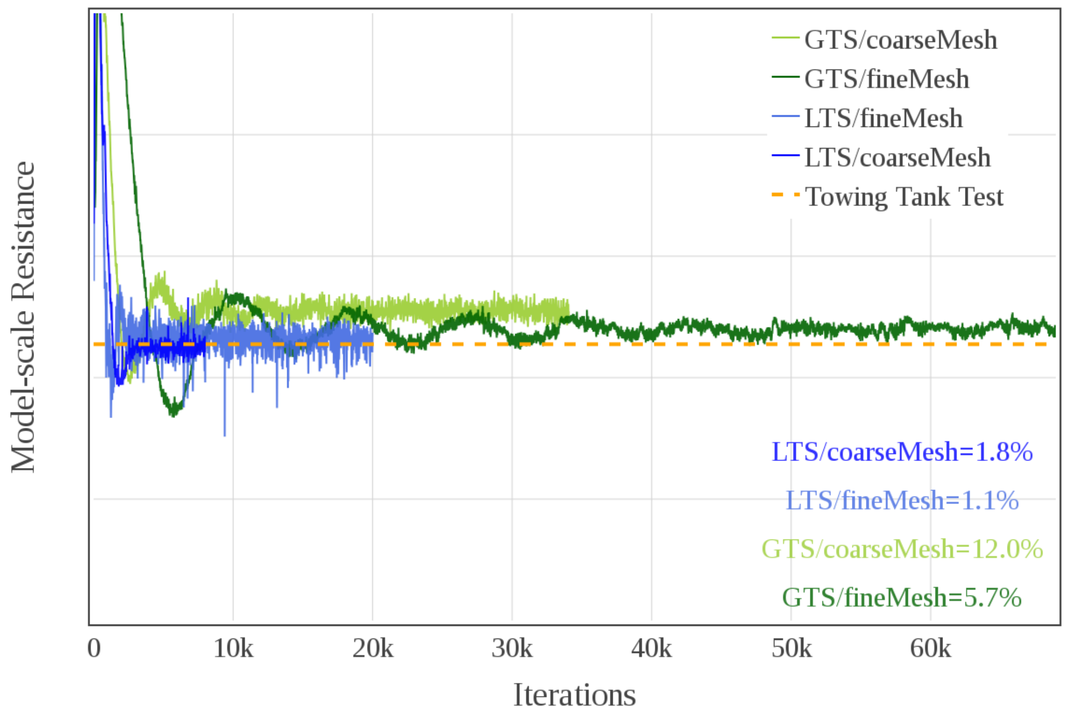


Figure 16.3: *Comparison between the convergence of total resistance simulation for four different numerical models: pseudo-steady (LTS) solution with a* 700k *cells mesh (blue), pseudo-steady (LTS) solution with a* 2.8M *cells mesh (light blue), unsteady solution with a* 700k *cells mesh (green) and a* 2.8M *cells mesh (light green). Errors with respect to the towing tank test value is reported at the bottom right of the plot. (Numerical values are not shown because data is confidential)*

Chapter 17

One Design Variable Optimization: protruding the bulb

A first approach to optimization has been to consider only one design variable. For this purpose the bulb's length parameter C_{LPR} has been chosen, since, as suggested in literature, it is one of the most important [10].

Thus the bulb has been morphed by displacing its nose along the longitudinal direction. Five displacements, from -0.1 to 0.3 meters with step 0.1, has been imposed. Giving them as input to MIMMO/MIMIC shape morpher built in *Chapter 13*, five geometries are obtained. In *Table 17.1* the linear parameters for the five

Nose Displacements [m]	-0.1	0.0	0.1	0.2	0.3
Length Parameter C_{LPR}	0.031	0.056	0.081	0.106	0.131
Breadth Parameter C_{BB}	0.160	0.170	0.175	0.178	0.181
Depth Parameter C_{ZB}	0.660	0.660	0.660	0.660	0.660

Table 17.1: *Nose displacements applied to the original geometry and correspondent values of the Kracht linear parameters.*

bulb configuration have been calculated as defined by Kracht [20]. Obviously the *depth parameter* C_{ZB} is the same for all geometries, since the morphing is made only by moving the nose in the *x-direction.* Also the *breadth parameter* seems not to vary significantly; slight variations are due to the fact that pulling the nose the wider part of the geometry comes forth, as could be seen in *Figure 17.1.*

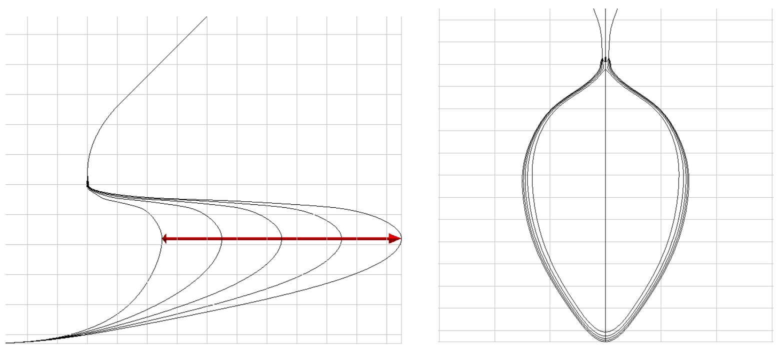

(a) *Profile of the five bulb configurations.* (b) *Effect of moving the nose on bulb's breadth at the Forward Point*

Figure 17.1: *Longitudinal (a) and traversal (b) sections of the morphed geometries.*

The behavior of these geometries is simulated using the OpenFOAM model set up in *Part III*. The purpose is to plot the trend of resistance with the protruding of the bulb. This have been done for three different full-scale speeds: 16, 18 and 20 knots ($Fn = 0.294, 0.331, 0.367$). Once the points are known a PYTHON script has been written to generate the surrogates.

In this chapter only polynomial regression is presented. The choice of the best order is analyzed and motivated. Furthermore, since some simulations have given noisy outcomes, an analysis is made and a way to deal with this problem is presented.

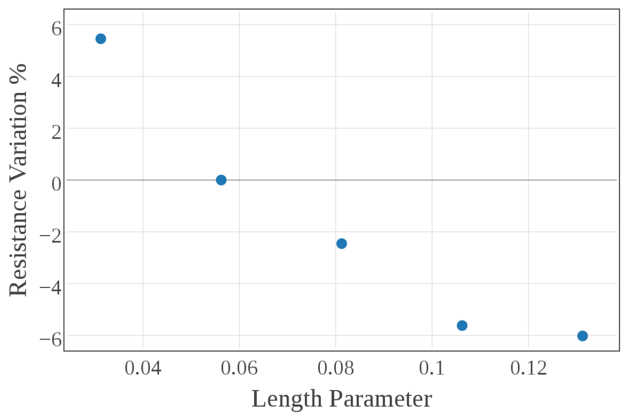
17.1 Catching the trend

This first optimization is a good example to show the process of surrogate modeling. From the OpenFOAM calculations three set of resistance values have been collected: these are the training data-sets for the three operative condition (*Figure 17.2*).

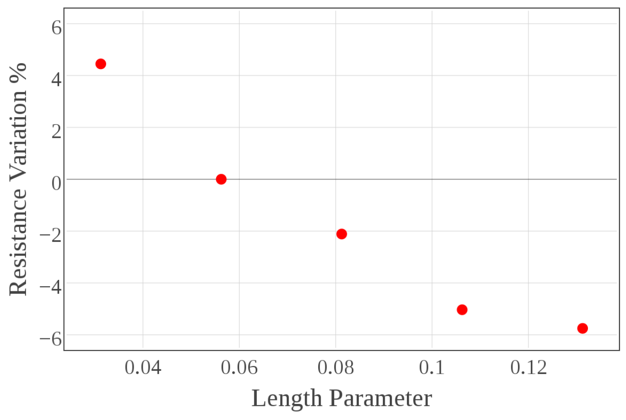
At this point, surrogate modeling comes into play. In *Chapter 15* the theories behind two surrogate modeling techniques have been introduced: *polynomial lest-squares regression* and *ordinary Kriging interpolation*. Based upon those mathematical formulation, a PYTHON script has been written to catch the trends described by the training data-sets.

First of all let's find the polynomial that most reasonably fits the training points. The first three order have been tried and results are compared in *Figure 17.3*.

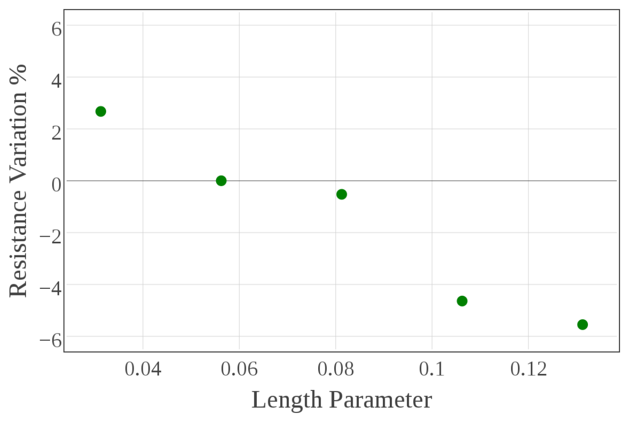
Looking at *Figure 17.3.a-b* it can be noticed that second and third order poly-



(a) *Training data-set at* $Fn = 0.294$

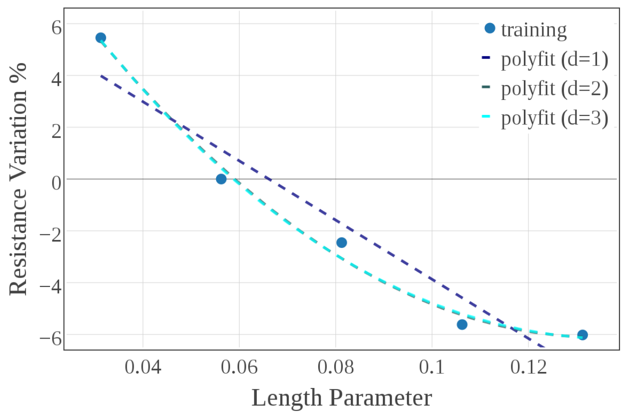


(b) *Training data-set at* $Fn = 0.331$

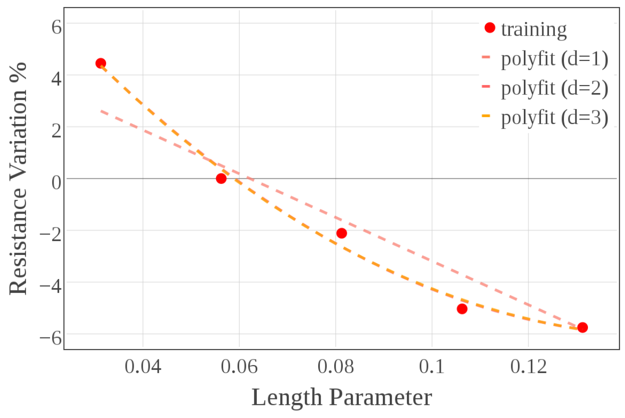


(c) *Training data-set at* $Fn = 0.367$

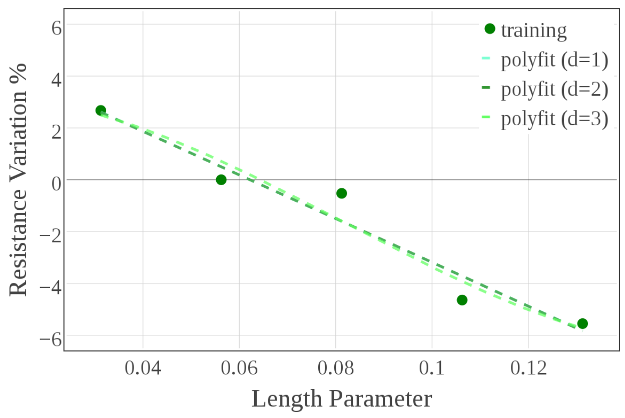
Figure 17.2: *Resistance variation from the undeformed case plotted against the bulb length parameter*



(a) *Polynomial fittings of training points at* $Fn = 0.294$



(b) *Polynomial fittings of training points at* $Fn = 0.331$



(c) *Polynomial fittings of training points at* $Fn = 0.367$

Figure 17.3: *Fitting of training data-sets using first, second and third order polynomials*

nomials coincide, while in *Figure 17.3.c* all the three functions overlap.

To help in the choice of the best polynomial degree a cross-validation can be used, but it hasn't been considered necessary in this case. Since, at a glance, the parabolic curve seems to give the more reasonable fit. This seems to hold only at the lower Froude regimes. In fact, at higher *Froude number*, the quadratic behavior is unable to deal with the presence of a central bump. This observation leads to an important question for the surrogate modeling: is this central bump a physical or a numerical consequence?

To answer this question an analysis of the simulation convergence "noise" has been made.

17.2 Evaluating the noise

One of the main principle adopted during the set up of the OpenFOAM experiment is the speed of calculus. The purpose has been the achievement of an enough accurate solution (compared with the experimental trend), with the minimum computational efforts. Once this setup was found, its dictionaries have been used for all the deformed geometries.

In *Figure 17.4* the convergence of total resistance is plotted for all the fifteen cases (5 geometries for 3 Fn). It can be noticed that some simulations reach convergence, but they are corrupted by a lot of noise (red and violet curves).

For all these results the mean value $\overline{R}$ is computed by averaging over the last 4000 iterations, where convergence is reached for all simulations. To measure the noise level of the simulations the percentage *standard deviation* σ from mean value is used:

$$N_\sigma = 100\frac{\sigma}{\overline{R}} \tag{17.1}$$

Noise amplitudes are collected in *Table 17.2* and plotted in *Figure 17.5*. *Figure 17.5*

shows an interesting feature of the training data-sets: the points that seems more departed from the 2-order regression curve correspond to a mean value extracted from noisier data. This doesn't necessarily mean that these data are wronger than others, nor that the chosen polynomial fitting is the reality, but it suggests that bumps or hollows in the training data-sets could derive from a bad averaging due to the noise.

This represents a problem in surrogate modeling, because leads to an uncertainty about some training points. If a pure *Kriging interpolation* was used, the surrogate

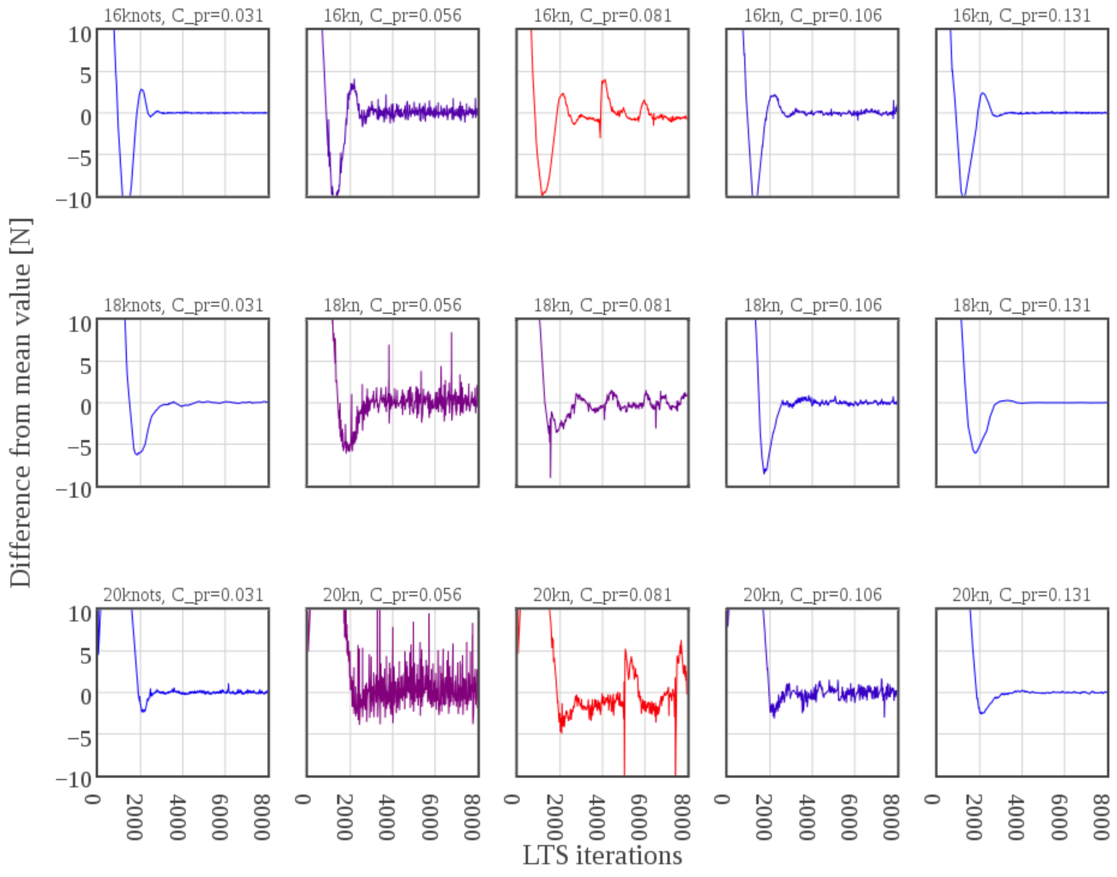


Figure 17.4: *Convergence of total resistance for all the cases (rows correspond to* $Fn = 0.294, 0.331, 0.367$*, columns to* $C_{LPR} = 0.031, 0.056, 0.081, 0.106, 0.131$*). The blue-to-red colors of the curves represent the numerical noise of the solutions (blue are smooth, red are noisy).*

model would have passed through all the training points, thus considering the bumps as a physical behavior. Since wave systems interference is involved this isn't an unfeasible idea, but it must verified in some way. In the following an attempt in doing so is presented.

17.3 Slightly modifying the schemes

This is not a numerical analysis of the *Local Time Stepping* technique, nor a comparison of different discretization schemes. The aim here is to show a way to dampen the noise and which consequences it brings on the results and on the surrogate modeling.

Fn / C_{PR}	**0.031**	**0.056**	**0.081**	**0.106**	**0.131**
0.294	0.07%	1.14%	3.43%	0.64%	0.14%
0.331	0.25%	1.63%	1.50%	0.40%	0.04%
0.367	0.17%	1.77%	3.33%	0.76%	0.11%

Table 17.2: *Noise amplitude of all the simulations studied in this chapter.*

Looking at the residuals it has been noted that the peaks during the convergence of the solution, correspond to peaks in the turbulence quantities' residuals. In this view it has been chosen to switch from a *Linear Upwind Differencing (LUD)* scheme, second-order accurate but oscillatory, to an *Upwind Differencing (UD)* scheme, less accurate (first-order) but bounded. This change dampens the numerical oscillations, by adding a streamwise numerical diffusion to the equations [14].

For example let's consider the case corresponding to the bump in the resistance-velocity trend, i.e. $C_{LPR} = 0.081$ and $Fn = 0.367$. In *Figure 17.6a* the convergence of the two different set up is compared.

As expected, switching to a single-order upwind scheme has introduced numerical diffusion, that bounds the solution by dampening the oscillations. This fact is highlighted by the distributions plotted in *Figure 17.6b*, where it is clear how the presence of oscillations lead to an overestimation of the resistance.

Substituting the new value in the training data-set, the fitting of the curves with a second-order polynomial becomes smoother and the bump disappear. However it must be remembered that a wave interference problem is considered, hence it is possible that certain lenghts lead to particular phenomena. So these oscillations could be a signal of an unsteady behavior that the model can't represent well enough and adding diffusion eliminates these results, hiding these features.

For a better understanding more detailed investigations have to be done, but this is left to future developments.

Concluding, in *Figure 17.8* the surrogates relative to the three *Froude Numbers* are compared. The parabolic trend seems the more reasonable to represent the decaying of resistance with increasing of the bulb's length. This means that probably there is a minimum of resistance for longer bulbs. However it must be reminded that the upper bound of the design space considered correspond to a bulb that is twice the original one and that its elongation is the 7.5% of the ship's L_{PP}; expanding the design space would be useless since probably it will be contradictory to the basic assumptions of the model.

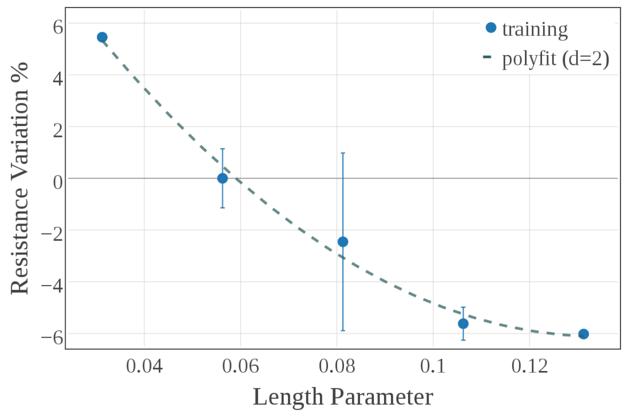
More interesting is the similarity of the three trends, suggesting that over these range of *Froude number* the resistance reduction with length behaves in the same manner.

17.4 Conclusions

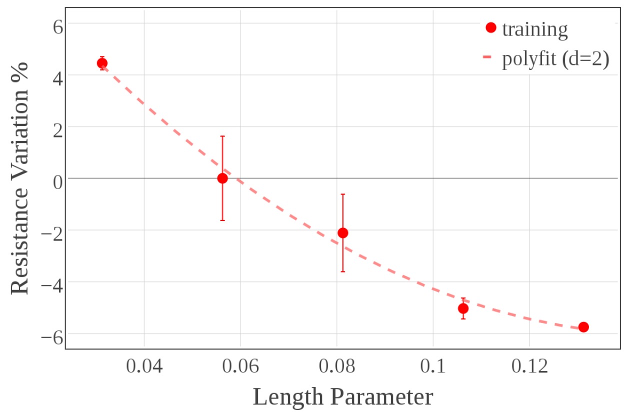
This chapter has presented a one design variable surrogate modeling, with the aim of studying the behavior of resistance while protruding the bulb. Following conclusions have been reached:

- the combined use of Dakota, MiMMO/MIMIC and OpenFOAM works well, giving good geometries and feasible results;
- the total resistance decays with the increasing of bulb's length following a parabolic trend;
- this trend seems to be the same for a range of *Froude Number* from 0.294 to 0.367, with the presence of bumps for higher values;
- the numerical model has a noisy outcomes for some of the simulated configurations, leading to errors in averaging the resistance. This could be responsible for the bumps in the resistance trend;
- to reduce the noise a strategy have to be found, for example in this work this has been done by switching from a *second-order* to a *first-order upwind* scheme for the turbulence variables;
- this change in the dictionary has led to an added numerical diffusion that dampens the oscillation;
- the surrogate built with the modified training data-set is smoother also at higher velocities, confirming the parabolic trend;
- a more detailed investigation must be done to confirm if the bumps at high velocities are physical or numerical;
- since there is only a decaying behavior, the optimum value is located at the upper limit of the design space;

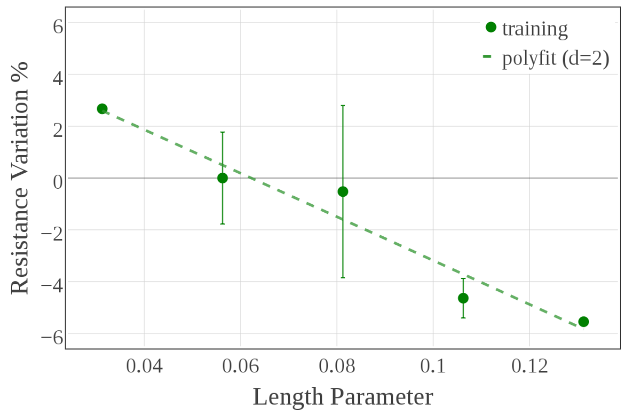
- parabolic trend makes thinking that for higher protrusion the benefit effect of protruding the bulb will vanish;
- in protruding the bulb, also ship design constraints must be considered in the optimization process, however this is left for future and more specialized developments.



(a) *2-order polynomial fittings of training points at* $Fn = 0.294$

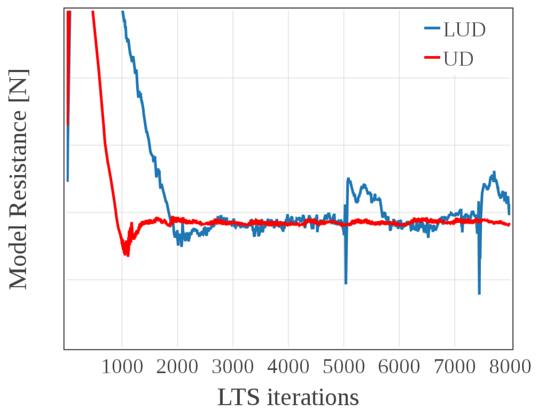


(b) *2-order polynomial fitting of training points at* $Fn = 0.331$

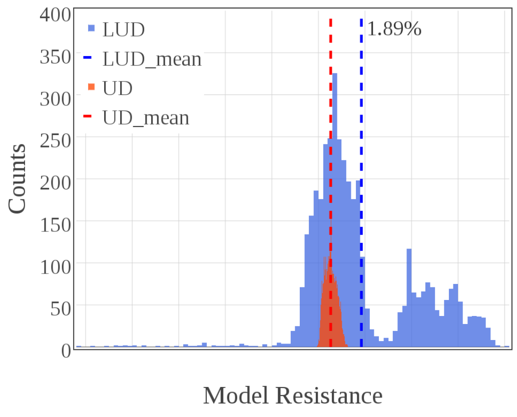


(c) *2-order polynomial fittings of training points at* $Fn = 0.367$

Figure 17.5: *Second-order polynomial fitting of the training data-sets with bar representation of the noise amplitude of each point.*



(a) *Convergence trends*



(b) *Simulation values distribution of the last 4000 iterations*

Figure 17.6: *Comparison between the simulated resistance values using a LUD or a UD scheme to discretize the turbulence quantities.*

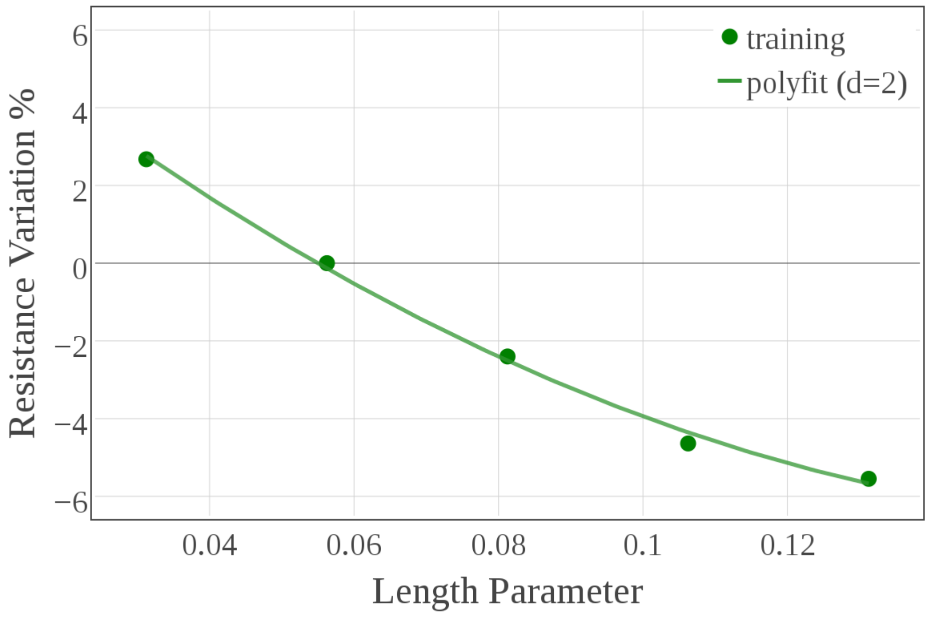


Figure 17.7: *Second-order polynomial fitting with the new training data-set.*

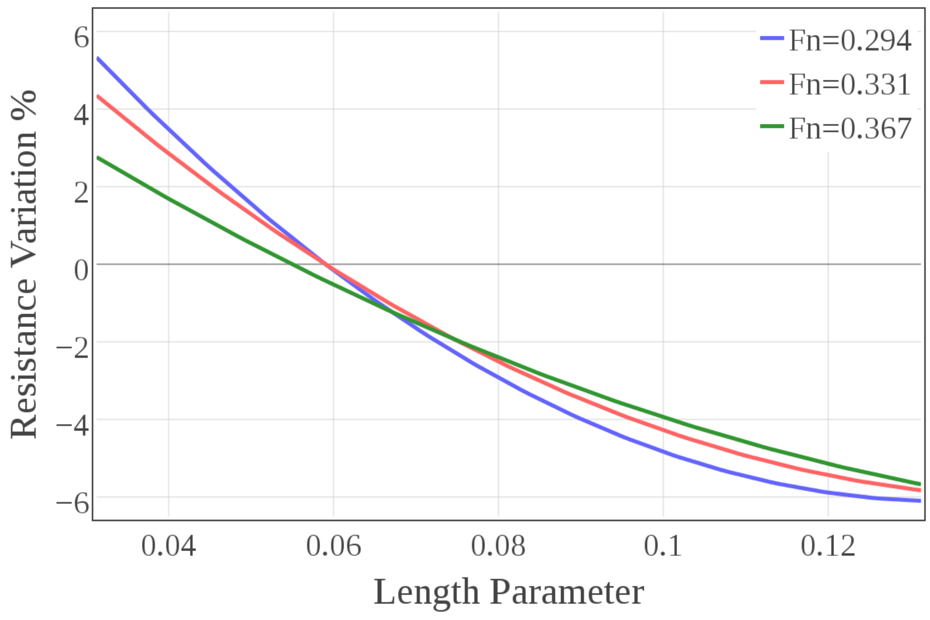


Figure 17.8: *Comparison of the resistance second-order polynomial surrogates constructed for* $Fn = 0.294, 0.331, 0.367$.

Chapter 18

Two Design Variable Optimization: displacing the nose

18.1 Displacements and Kracht Parameters

In shape morphing problems there are a lot of variables that can be varied to find the optimal configuration. Including more variables in the optimization problem statement augment the procedure. However these are the cases in which the power of surrogate modeling comes into play.

In the previous chapter a first approach to surrogate modeling has been presented. Only one design variable, the *length parameter* C_{LPR}, has been considered and a surrogate model, i.e. a curve, has been constructed to represent the physical trend.

In this chapter another design variable is added to the problem: the bulb depth parameter C_{ZB}. This means that the nose of the bulb is displaced not only in the *x-direction*, but also in the *z-direction*.

Three different position along z has been imposed, that, replicated for all the five x displacements of the previous chapter, leas to a grid of fifteen points.

Giving these values to the MiMMO/MIMIC shape morpher, fifteen geometries have been obtained. The Kracht linear parameters relative to these configurations are collected in *Table 18.1* .

The *breadth parameter* hasn't been shown since it has the same value presented in the previous chapter and it isn't considered in the study. Geometries described by the combinations of these parameters are sketched in *Figure 18.1.*

Behavior of this geometry is simulated using the OpenFOAM model set up in *Part III.*

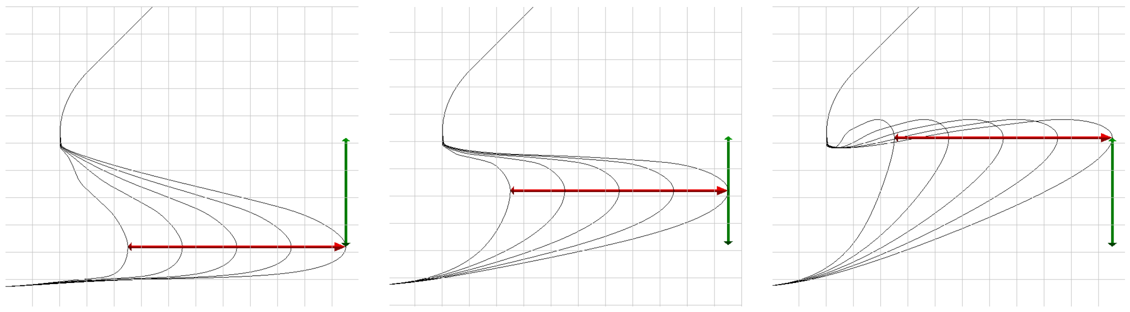
X [m]	-0.1	0.0	0.1	0.2	0.3
$\boldsymbol{C_{LPR}}$	0.031	0.056	0.081	0.106	0.131

(a) *Length Parameter*

Z [m]	$\boldsymbol{C_{ZB}}$
0.1	1.038
0.0	0.660
-0.1	0.283

(b) *Depth Parameter*

Table 18.1: *Nose displacements along x and z applied to the original geometry and correspondent values of length and depth parameters.*



(a) *Profile of the five bulb configurations with* $C_{ZB} = 0.283$. (b) *Profile of the five bulb configurations with* $C_{ZB} = 0.660$. (c) *Profile of the five bulb configurations with* $C_{ZB} = 1.038$.

Figure 18.1: *Longitudinal sections of the morphed geometries.*

As for the one-design-variable case, the aim has been to understand the effects of the bulb's deformation on the total resistance. From the previous analysis it has been found that the protruding of the bulb leads to a parabolic decaying of the hydrodynamic drag. Now, the effect of the vertical position of the bulb's nose wants to be investigated.

In this case the surrogate is no more a curve, but a *surface*. A more accurate and reliable surrogate constructor is used, so DAKOTA toolkit substitutes the in-house PYTHON script.

Again, the purpose is to construct this response surface for three different full-scale speeds: 16, 18 and 20 knots ($Fn = 0.294, 0.331, 0.367$). The three training data-sets obtained with OPENFOAM are shown in *Figure 18.2*

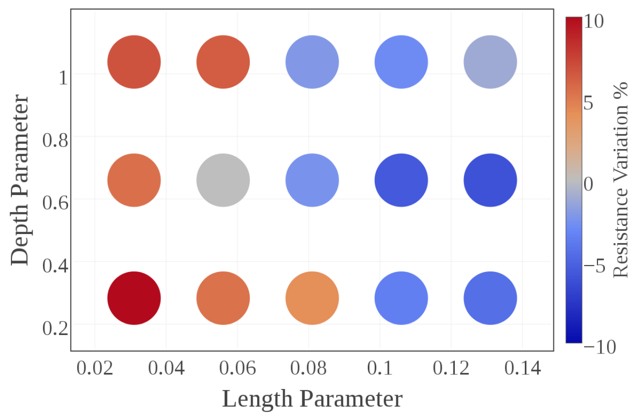
Before going into the surrogate construction, it is worthy to make a comment about the ship's volume variation during shape morphing. The volume displacement represents one of the main ship design parameter, so with a view to optimize the bulb for the same ship, this value should be kept constant. To do this a more sophisticated set up of the shape morpher should be made. However, the deformations considered in these work lead to small variations of the underwater volume of the ship. So, for the purpose of this thesis the volume variations have been considered negligible. A more controlled and constrained shape morphing procedure is suggested for future developments.

18.2 Building the Surrogates

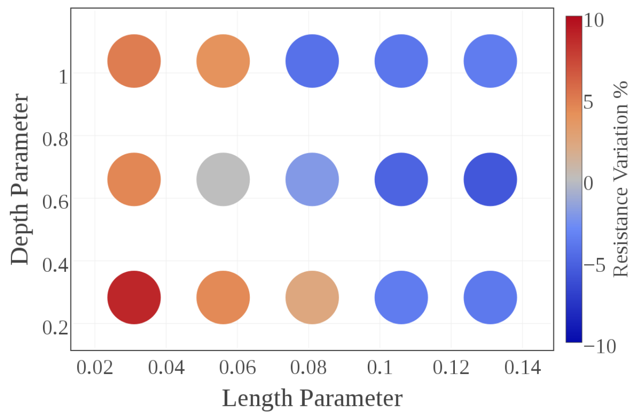
According to the conclusions made in the previous chapter, the first surrogate model tried is a second-order polynomial. So, by giving the training data-sets as input to DAKOTA and by specifying the domain where the surrogate must be constructed ($C_{LPR} = [0.02, 0.14]$ and $C_{ZB} = [0.2, 1.1]$), the response surfaces have been built. These are plotted in *Figure 18.3*.

Looking at these contours it seems that lifting and lowering the nose doesn't lead to a resistance reduction, rather it seems to slightly worsen the performance. This aspects seems to be enhanced for longer bulbs. However these results must be analyzed with care. In fact the polynomial fitting works well in the case of proptruding the bulb, but the influence of the nose's depth it is unknown.

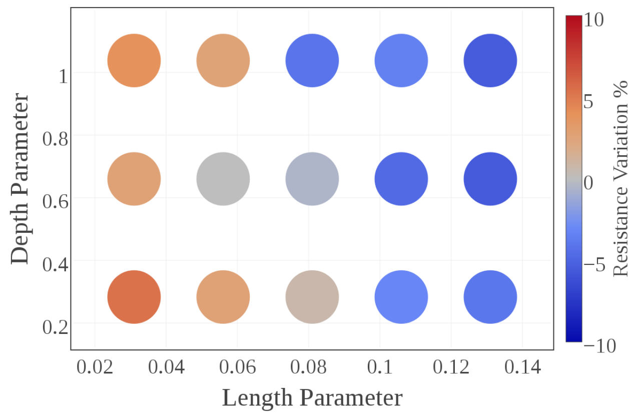
Polynomial regression doesn't pass through the training points, but try to minimize the distances from them. This can help to exclude the noise, but it can also hide important features of the flow.



(a) *Training data-set for* $Fn = 0.294$

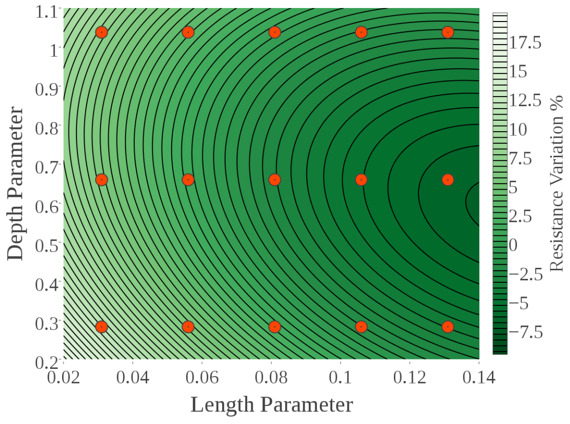


(b) *Training data-set for* $Fn = 0.331$

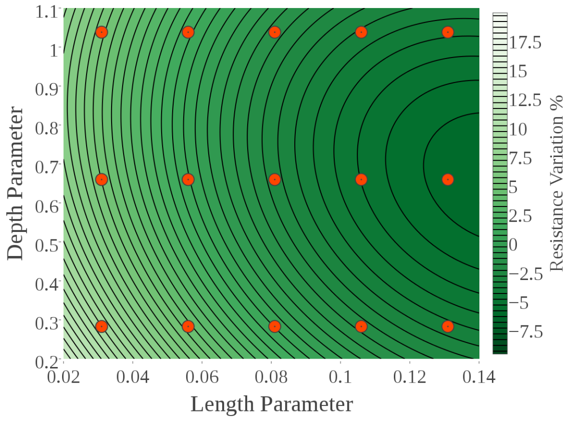


(c) *Training data-set for* $Fn = 0.367$

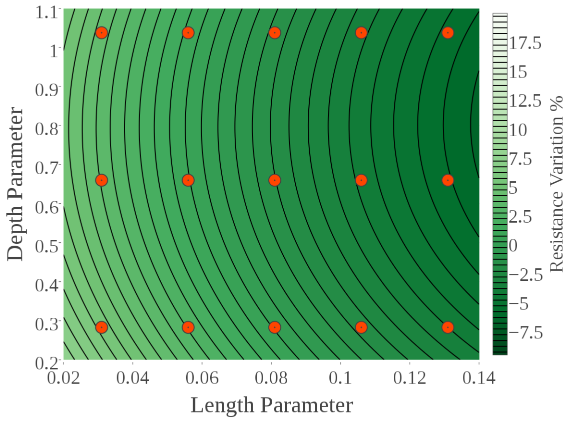
Figure 18.2: *Training (●) data-sets at three different Froude number. Blue-Red colorscale represents the percentage resistance variation from the original bulb configuration.*



(a) *Contour of the polynomial response surface at* $Fn = 0.294$



(b) *Contour of the polynomial response surface at* $Fn = 0.331$



(c) *Contour of the polynomial response surface at* $Fn = 0.367$

Figure 18.3: *Second-order polynomial surrogate models of percentage resistance variation from undeformed for each length/depth configuration and for each tested Froude number.*

For this reason another common and very efficient technique has been used: the Kriging interpolation. This method predict the new points by interpolating the training data-set, so the response surface will pass trough the high-fidelity points. Results obtained in this manner are shown in *Figure 18.5*.

Comparing the contours in *Figure 18.3* with those in *Figure 18.5* an important difference becomes visible. The 2-order polynomial regression gives very smooth surfaces, while the surface generated by Kriging process are less smooth. This difference is due to the fact that Kriging surrogates are constrained to the training points and hence if some physical behaviors or the presence of numerical noise create bumps in the solutions, Kriging will catch them..

To understand which one is the best representation of the ship' s resistance fifteen design points couldn't be enough. So before making a comparison between the two models, it is necessary to test the surrogates. This has been done only for the low velocity case ($Fn = 0.294$).

18.3 Testing the surrogates

Response surfaces have been constructed using two different approaches: 2-order polynomial and Kriging. These models have been computed from 15 training points. This is a poor training-set, usually more points are required. Thus they need to be tested and validated before making an in-depth optimization study.

In the context of this thesis the validation is focused on the case at $Fn = 0.294$. This surrogate is tested using 10 new design points, relative to two intermediate z-position ($z = -0.05, +0.05$) of the bulb's nose. Parameters of the ten new configurations are collected in *Table 18.3*.

X [m]	-0.1	0.0	0.1	0.2	0.3
C_{LPR}	0.031	0.056	0.081	0.106	0.131

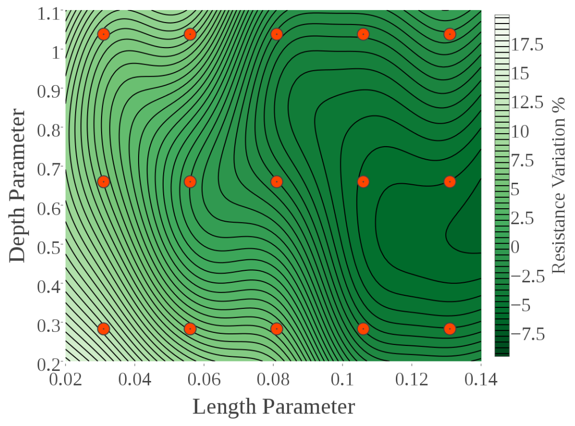
(a) *Length Parameter*

Z [m]	C_{ZB}
0.05	0.849
-0.05	0.472

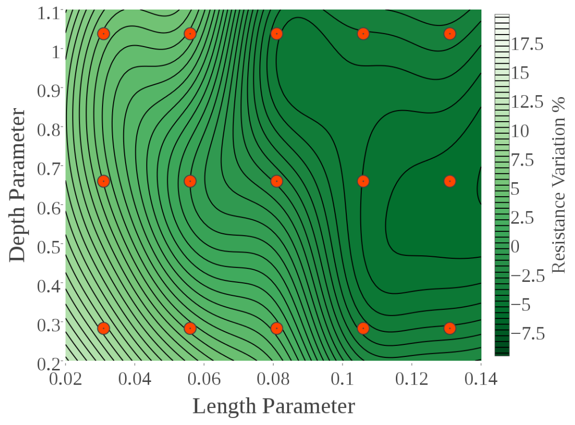
(b) *Depth Parameter*

Table 18.2: *Design variable of the test data-set. In table (a) there are the nose displacements along x and the correspondent length parameters, in table (b) there are the nose displacements along z and the correspondent depth parameters.*

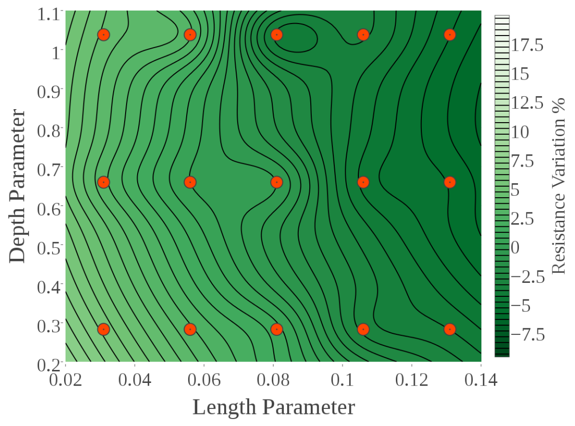
Results of the OpenFOAM simulations for the test data-set are shown in *Figure*



(a) *Contour of the Kriging response surface at* $Fn = 0.294$

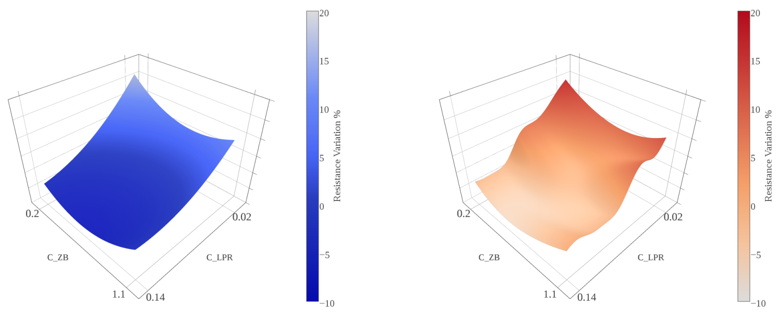


(b) *Contour of the Kriging response surface at* $Fn = 0.331$



(c) *Contour of the Kriging response surface at* $Fn = 0.367$

Figure 18.4: *Kriging surrogate models of percentage resistance variation from undeformed for each length/depth configuration and for each tested Froude number.*



(a) *2-order polynomial* (b) *Kriging*

Figure 18.5: *Response surfaces generated for* $Fn = 0.294$ *with two different approach: 2-order polynomial (a) and Kriging (b).*

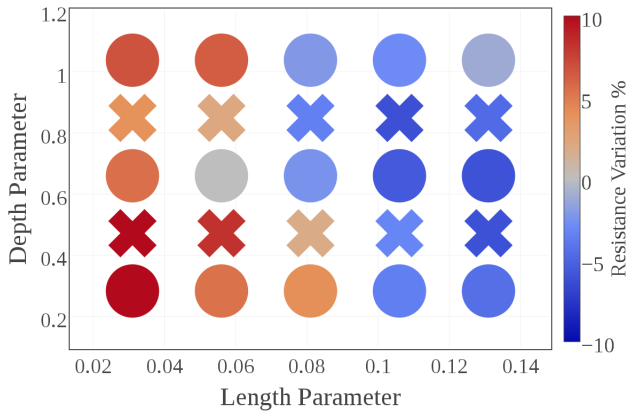


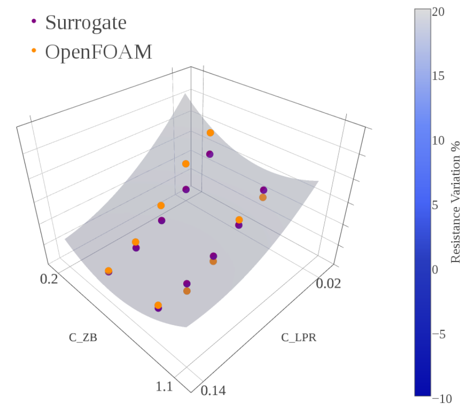
Figure 18.6: *Training* (•) *and test* (×) *data-sets at* $Fn = 0.294$. *Blue-Red colorscale represents the percentage resistance variation from the original bulb configuration.*

18.6.

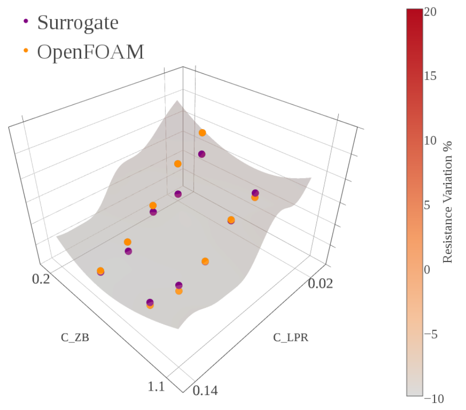
The testing values are then compared with the surrogates prediction for those design variables combination. The differences can be seen in *Figure 18.7.*

The number of training points isn't enough to catch the resistance trend. Errors in predicting the percentage resistance variation reach unacceptable values. However, looking at *Figure 18.8* it can be noticed that these mispredictions are concentrated in the zone of short and deeply immersed bulbs, while the differences are less significant in the minimum zone previously identified.

In all the three surrogates it seems that the up-and-down displacement of the nose has a unimodal effect on resistance, corresponding to a central "valley" along the x-direction. But, the test data-set suggests the presence of bumps and hollows in

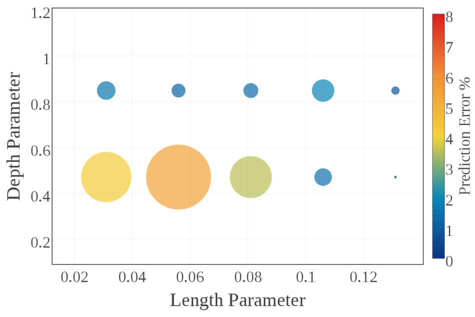


(a) *2-order polynomial*

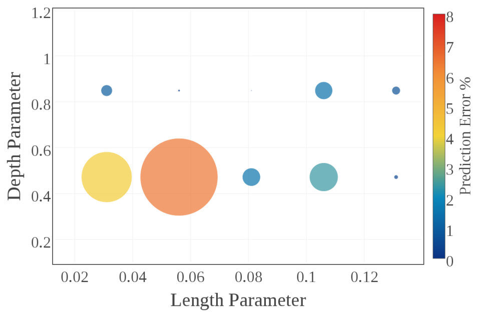


(b) *Kriging*

Figure 18.7: *Comparison between surrogate predictions and* OpenFOAM *results at* $Fn = 0.294$*. The two figures refer to: 2-order polynomial (a) and Kriging (b).*



(a) *2-order polynomial prediction errors*



(b) *Kriging prediction errors*

Figure 18.8: *Error in surrogate prediction in comparison with the* OpenFOAM *resistance value at* $Fn = 0.294$*. The two figures refer to: 2-order polynomial (a) and Kriging (b).*

the region of short protruding length. This effect is present also at high protrusion, but it is smoother.

At this point, the test data-set can be included in the training data-set to construct a new surrogate. This procedure is commonly called "infilling".

18.4 Infilling and Comparing

By adding the test points to the training data-set, the number of design points has been increased from 15 to 25. So the new training data-set will be composed by the combinations of the parameters collected in (*Table 18.3*).

X [m]	-0.1	0.0	0.1	0.2	0.3
C_{LPR}	0.031	0.056	0.081	0.106	0.131

(a) *Length Parameter*

Z [m]	C_{ZB}
0.1	1.038
0.05	0.849
0.0	0.660
-0.05	0.472
-0.1	0.283

(b) *Depth Parameter*

Table 18.3: *Nose displacements along x and z applied to the original geometry and correspondent values of length and depth parameters.*

Resistance values are computed for this new configurations and the new training data-set is obtained. These values are shown in *Figure 18.9*.

On the basis of this more complete set of training points two new surrogates are built: the first with a second-order polynomial regression, the second with a Kriging interpolation. These two new surfaces are shown in *Figure 18.10*.

As can be seen by the contour plots, both approaches lead to a similar trend, but they highlight some different features.

The common feature is the presence of a minimum zone for long bulbs with nose at an intermediate depth. While the polynomial trend tells that the only minimum is beyond the higher bound of the length design variable, Kriging generates a local minimum inside the design space.

The main difference is in the zone of shorter bulbs: the polynomial presents one big "valley" for each length parameter, while the interpolating approach, instead,

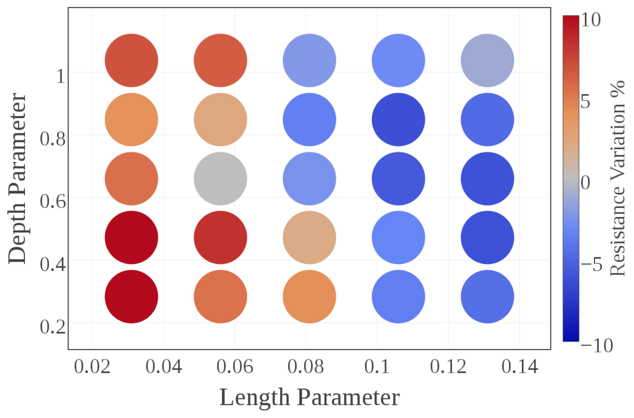
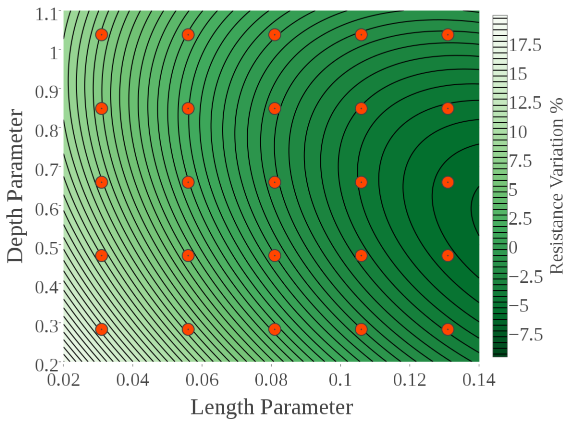
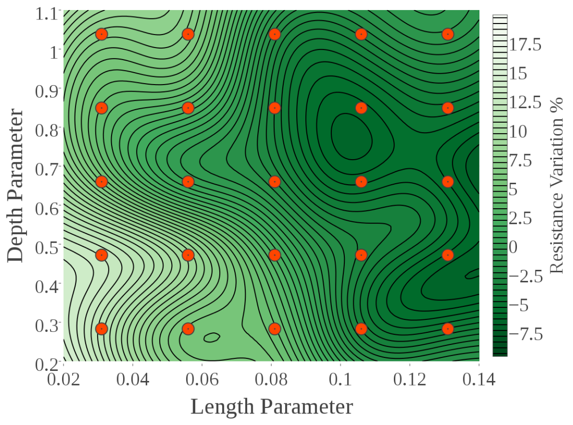


Figure 18.9: *Training data-set of percentage resistance variations from undeformed for each length/depth configuration tested at $Fn = 0.294$.*



(a) *Contour of the 2-order polynomial response surface*



(b) *Contour of the Kriging response surface*

Figure 18.10: *Kriging surrogate models of percentage resistance variation from undeformed for each length/depth configuration and for each tested Froude number. In this case the training data-sets is composed of 25 design points.*

generates bumps and hollows around the design space.

Kriging approach offers a way to relax its interpolating nature, by allowing the surrogate not to pass through the training points. This could be very useful in the cases where the numerical noise could corrupt some resistance values.

This is done by adding a "nugget" term in the diagonal of the correlation matrix. This correction leads to the response surface presented in *Figure 18.11*.

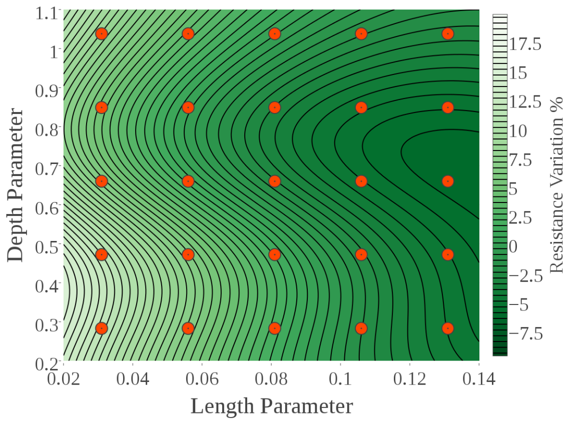


Figure 18.11: *Results of Kriging interpolation with the addition of a "nugget" term.*

This third surrogate seems to be a good compromise between the previous ones, since it can consider the physical feature of the flow while limiting the risk of over-fitting the noise.

18.5 Validating

At this point a surrogate must be chosen. Three response surfaces have been constructed using 2-order polynomial regression, Kriging interpolation and Kriging with "nugget". These are built upon a 25 design points data-set. This training data-set seems to be a good sampling, being able to catch some features of the bulb's shape effect on resistance, especially for the depth parameter.

To choose the best surrogate another test data-set is used. This consists of three new design points located in the minimum zone. Parameters representing these three configurations are presented in *Table 18.4*.

The test resistance values are computed with OpenFOAM and the percentage resistance reduction calculated. In *Table* **??** these values are compared with the surrogate predictions.

Results show the augment of accuracy obtained by passing from a 15 to a 25 points training data-set. Overall, the best method seems to be the Kriging inter-

C_{LPR}	C_{LPR}
0.119	0.660
0.131	0.566
0.131	0.755

Table 18.4: *Bulb's length and depth parameters for the new three configurations. These data-pairs are the new test data-set.*

C_{LPR}	C_{LPR}	**poly2**	**Kriging**	**Nugget**	**OpenFOAM**
0.119	0.660	-5.03	-5.77	-6.14	-6.17
0.131	0.566	-5.05	-6.48	-6.12	-6.04
0.131	0.755	-6.11	-6.07	-5.73	-6.65

Table 18.5: *Resistance reduction given by test configurations calculated with* OpenFOAM *and predicted with 2-order polynomial, Kriging and Kriging with nugget surrogates.*

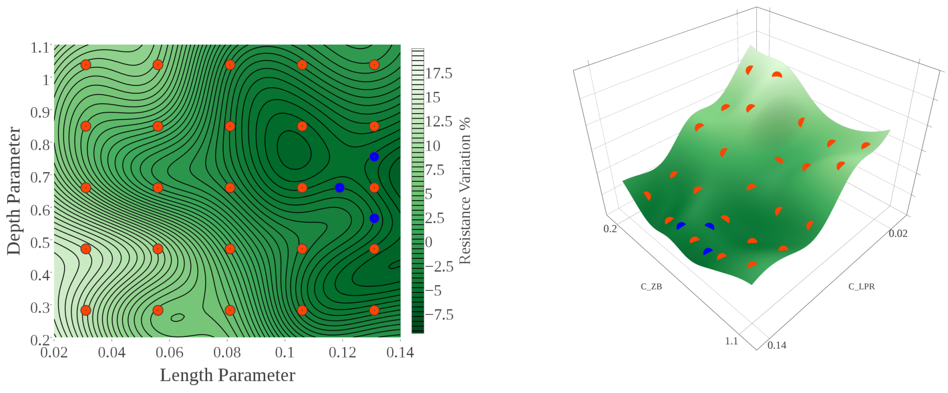
polation without nugget, but there are slight differences from other two methods. However, it must be stressed that this results refer only to the minimum zone of the design space. To choose the globally best method further test points should be defined across the design space.

For the purpose of this thesis the Kriging method is chosen has surrogate approach. The new test-set is added to training set and, again, a new surrogate is obtained. This response surface is plotted in *Figure 18.12*.

Compared to the first surrogates obtained (see the previous sections), this final response surface has more curvy behavior, presenting more than a valley. Again, it should be verified in detail if the bumps are numerical or physical, or a combination of them, but this is left as a further development.

This model can be considered a good representation of the numerical resistance outcome and, within the limits of the simulation, also of the ship hydrodynamics. The choice of using a light numerical model has allowed to count on a substantial number of training points, but inevitably it brings noise and inaccuracy to the surrogate.

During this chapter, all the main stages of surrogate modeling has been pre-



(a) *Contour plot* (b) *3-D surface*

Figure 18.12: *Contour (a) and three-dimensional surface (b) representations of the surrogate model at* $Fn = 0.294$*. The surface is built applying Kriging interpolation to 28 training points (orange and blue). Infilling is made with 3 points (blue) in the minimum zone.*

sented. Firstly, a training data-set consisting of 15 design points is tried. Three different techniques (2-order polynomial, Kriging, Kriging with nugget) are used to generate the surrogates. These surrogates have then been tested with 10 new design points. The representations show the global trend, but are not enough to catch the effect of depth.

The test points are added to the training data-set and three new surrogates are built, leading to a better description of the resistance behavior. To understand which one is better a new test data-set has been defined in the minimum zone. This second test shows that the three models have about the same accuracy in the zone of interest, but among them the Kriging seems to give the better predictions.

Again test points are added to training data-set and the surrogate is reconstructed. The result is a more detailed and curvy surface.

This procedure should be repeated many times to achieve a very detailed surrogate. However, it is thought that the one achieved is enough for the purpose of this optimization. At this stage, instead, efforts should be directed toward an accurate numerical solution to understand in detail the physics of the phenomena, but this is left as a further improvement.

Chapter 19

An approach to robust optimal solution

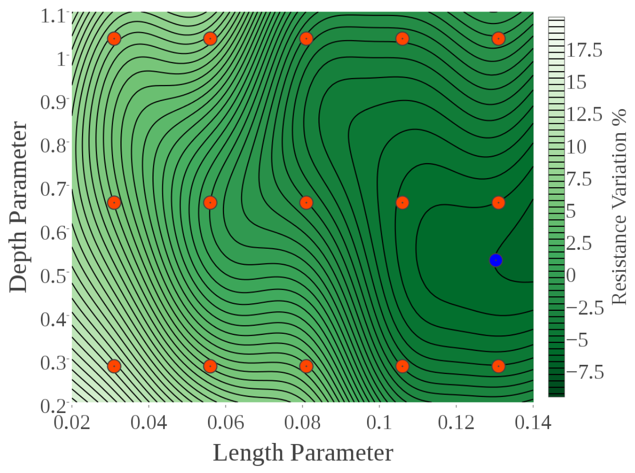
The construction of the surrogate has been made considering only two design variables: the length and the depth parameters. This has been done for three different operating condition $Fn = 0.294, 0.331, 0.367$, corresponding to $16, 18, 20$ knots of full scale speed. Globally, the velocity doesn't seem to affect significantly the percentage resistance variation (from undeformed configuration). However, in the response surfaces the minimum (inside the design space) seems to slightly change its position.

This means that a bulb optimized for a certain Fn regime won't be the optimum at another sailing speed. In this view it is interesting to present an approach to find a more robust optimal solution. Without entering the branch of Robust Optimization, which is a specialized analysis, in this chapter the word robust is used to describe a bulb configuration that gives the optimal performance over a range of different operating conditions.

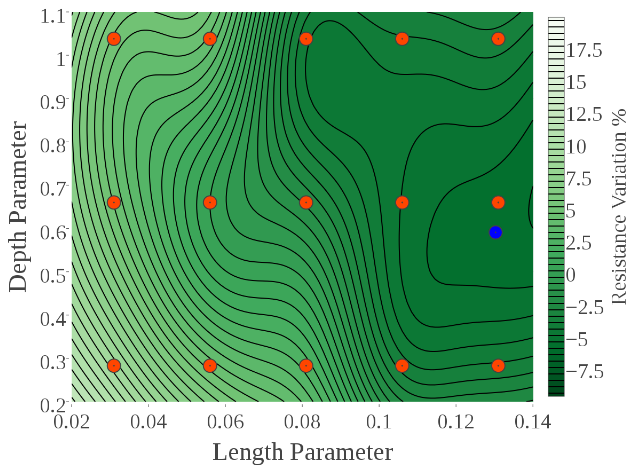
It must be reminded that the assumptions at the base of the numerical model become inappropriate for high speed values, when the effects of dynamic trim and hydrodynamic phenomena could become significant. A more detailed analysis of the effect of velocity should consider a wider range of Froude Numbers. Nevertheless, the purpose of the work presented here has been the creation of an optimization procedure. In this view, it becomes interesting to study the effect of velocity as an example of the possibilities given by the approach developed.

The Kriging surfaces have been chosen to investigate the effect of Fn on the optimum configuration, since they present the more "curvy" behavior.

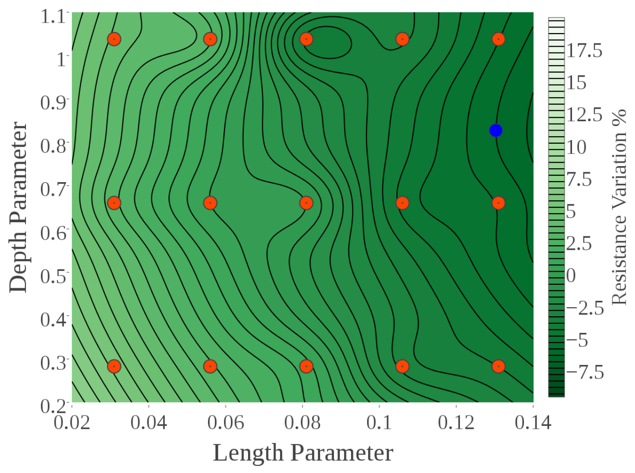
Figure 19.1 highlight three main features:



(a) *Kriging surrogate for* $Fn = 0.294$



(b) *Kriging surrogate for* $Fn = 0.331$



(c) *Kriging surrogate for* $Fn = 0.367$

Figure 19.1: *Kriging surrogate models of percentage resistance variation from undeformed for each length/depth configuration and for each tested Froude number. The orange points are the training data-set (15 points) and the blue ones correspond to the minimum in the limits of the design space.*

- the protrusion is the predominant factor in diminish the resistance;
- the optimal nose's immersion is around the middle point between the free surface and the bottom of the hull;
- velocity seems to move the optimal configuration towards long and shallow bulbs;

The minimums have been taken inside the design space, not to violate the model assumptions. Furthermore, too long bulbs probably are not acceptable for ship's design constraints. In *Table 19.1* the three optimal configuration are presented.

$\boldsymbol{Fn}$	$\boldsymbol{C_{LPR}}$	$\boldsymbol{C_{ZB}}$
0.294	0.131	0.527
0.331	0.131	0.591
0.367	0.131	0.827

Table 19.1: *Bulb's parameter for the optimal configuration at three different Froude number. The values are predicted with Kriging on a 15 design points data-set.*

The three optimal configurations have been simulated in OpenFOAM. The results of these simulation are plotted in *Figure 19.2* .

Again, the values of mean resistance are computed by averaging over the last 4000 iterations. The percentage resistance reduction from the original configuration is calculated and compared with the surrogate prediction (*Table 19.2*).

$\boldsymbol{Fn}$	$\boldsymbol{\Delta R}\%$ **Kriging**	$\boldsymbol{\Delta R}\%$ **OpenFOAM**
0.294	-6.51	-4.75
0.331	-5.80	-2.99
0.367	-6.00	-5.74

Table 19.2: *Comparison between the optimal resistance reduction predicted by Kriging surrogate and computed with* OpenFOAM.

Error of predictions are very high for $Fn = 0.294$ and $Fn = 0.331$, while are very small for the higher case. An interpretation of this result can be made considering

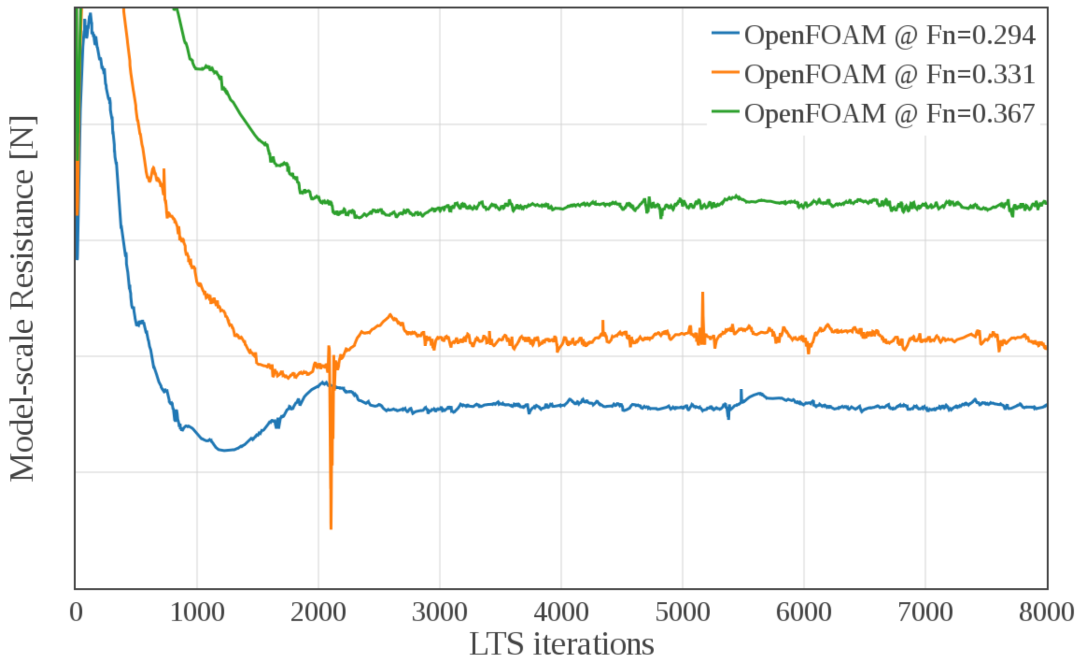


Figure 19.2: OpenFOAM *resistance calculations for the optimal configurations at* $Fn = 0.294, 0.331, 0.367$. *(Numerical values are not shown because data is confidential)*

the test data-set. In fact, as it has been noted in the previous chapter, a training data-set of 15 design points seems not to be enough to construct a reliable surrogate. In particular it has been pointed out the presence of sinuous trend with the depth parameter, especially for shorter bulbs. This effect could be responsible for the prediction errors, locating a minimum where instead there is a bump.

However, regarding the aim of this section, this minimum are considered acceptable.

A way to consider the effect of sailing speed on the optimum configuration is to compute an average of the three surrogates. This can be made by computing an average or a weighted average on the three Froude numbers. The formula used is:

$$R_{avg} = \frac{c_{16}R_{16} + c_{18}R_{18} + c_{20}R_{20}}{c_{16} + c_{18} + c_{20}} \tag{19.1}$$

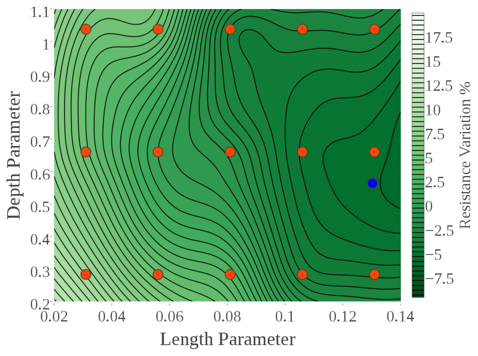
where $R_{16,18,20}$ are the resistance calculated at $Fn = 0.294, 0.331, 0.367$ and $c_{16,18,20}$ are the weights of these regimes.

If all the three regimes have the same importance, then $c_{16} = c_{18} = c_{20} = 1$ (*Figure 19.3.(a-b)*), otherwise if some regimes are preferred the weights can be changed. For example, let's imagine that the ship is designed to travel most of the time at 20 knots ($c_{20} = 2$), at 16 knots during the maneuvering ($c_{16} = 0.5$) and at

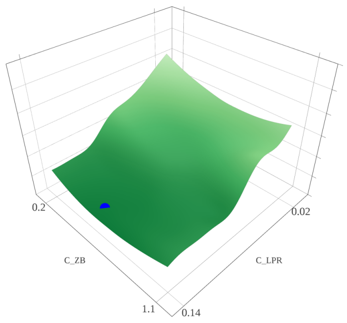
18 knots ($c_{18} = 0.25$) only to reach the design speed. This situation is plotted in *Figure 19.3.(c-d)*.

Optimal configurations at $Fn = 0.294$ and $Fn = 0.331$ are very similar (*19.1*) then, if a classical average is chosen, optimal configuration will resemble those, presenting a more immersed bulb. Instead, if more relevance is given to the higher Froude, the optimal configuration will tend toward shallow bulbs.

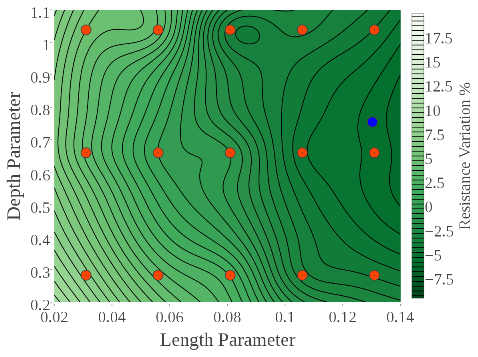
These considerations must be taken with care because the surrogates used are built on a poor training data-set. To develop a more consistent analysis, surrogates must be tested and validated (as it has been done in *Chapter 18*). However, the purpose of this chapter was to give an hint on how to consider the sailing speed in optmizing the bulb's shape.



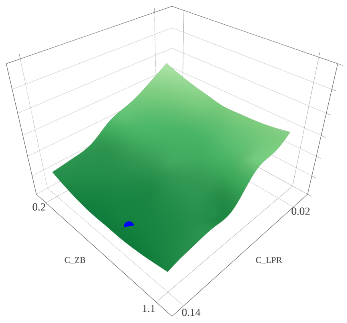
(a) *Contour plot*



(b) *3-D surface*



(c) *Contour plot*



(d) *3-D surface*

Figure 19.3: *Contour and three-dimensional surface representations of the Froude-averaged surrogate model. Average has been made giving all the regimes the same weight (a-b) or favoring the higher Froude number (c-d).*

Conclusions and Future Developments

Shape optimization often requires expensive analysis both in experimental tests (test set up, construction of prototypes, etc.) and in numerical simulations (expensive software licenses, computational resources). In this context an optimization framework based on open source tools and on effective numerical simulations has been developed.

This framework has been validated with the practical application to the bulb of a ship provided by *Fincantieri S.p.A.*. In particular interests have been focused on studying the effect of protrusion and immersion of the bulb's nose.

In this view a shape morpher has been constructed with the MiMMO library package developed by *OPTIMAD Engineering srl.* This tool is able to generates new accurate geometries by imposing displacements to the bulb's nose. In this manner it has been possible to vary independently two parameters of the bulb (C_{LPR} and C_{ZB}).

The resistance of each bulb geometry has been calculated by an OpenFOAM numerical model. Results of this model has been validated by benchmark with experimental results and with a finer (but much more expensive) numerical model.

The optimization strategy has been based on surrogate modeling, with the aim to build a response surface for two design variables: the length and depth bulb parameters (C_{LPR} and C_{ZB}).

A first approach has been explored considering only the effect of protrusion at three different Froude regimes ($Fn = 0.294, 0.331, 0.367$), corresponding to $16, 18, 20$ knots of full-scale speed. This analysis has shown a parabolic decaying behavior of the resistance with the increasing of bulb's length.

For the higher Froude number ($Fn = 0.367$) a central bump seems to contradict the quadratic trend observed at lower Froude number. The presence of numerical noise in the corresponding simulation has suggested an analysis of the discretization schemes. As an example the discretization scheme of the turbulent quantities has been changed from a second to a first order. This adds numerical diffusion that dampens the oscillations, leading to a smoother output. This change has lowered the value of resistance, making the bump disappear.

Over the Froude range considered it seems to be no differences in resistance reduction with protrusion, hence the bulb, as long as the trim variations are negligible, probably will behave the same at different velocities.

After that, the effect of the nose immersion has been investigated. This has been done with a two-design-variable analysis.

A full-factorial sampling of 15 design points (3 immersions and 5 lengths) has been chosen and the corresponding training data-set has been used to build the surrogates for the three Froude regimes. Surrogate building methods that have been used are second-order polynomial regression and Kriging interpolation. Both have given similar results, detecting the beneficial effect of protrusion and intermediate nose immersions.

These response surfaces have been validated using a test data-set of 10 points (2 intermediate C_{ZB} for each C_{LPR}). This has shown the unreliability of the 15 points training data-set, especially for short and deeply immersed bulbs. So a more detailed surrogate has been constructed by including the test points in the training data-set. This has been done only for the case of low velocity.

Second-order polynomial regression gives more or less the same results, while the Kriging gives curvier surfaces. This is due to the inherent nature of the two models: interpolation and regression. For this reason an in intermediate approach has been tried, allowing the Kriging not to pass through the training points (”nugget” term addition).

The three response surfaces have been tested to understand which one gives the best predictions. This has been done defining three test points in the zone of interest, i.e. the minimum zone (long bulb with intermediate depth). All the models give good predictions, but among them the Kriging provides slightly better results.

Again the training data-set is infilled with the test points to generate a new surrogate (using the Kriging). This procedure could have been repeated many times and in other zones, but the response surface obtained has been considered accurate enough in predicting the outcomes of the numerical model, especially in the minimum zone.

Finally, an approach to consider the effect of velocity (Froude) in choosing the best bulb configuration has been tried. This has been done by taking the Kriging models constructed with the coarser training data-set (15 points) and by averaging the value of resistance for the three different Froude regime. Weights can be used in the average in order to privilege certain regimes more than other.

The work that has been presented in this thesis has led to the construction of an open source optimization framework. However, the practical application to a naval problem has shown only some of its capabilities.

Regarding the bulb shape optimization more detailed investigations have to be done in order to be able to find the feasible optimal configuration. This will have to involve essentially two things: a more complete parameterization of the bulb and an improvement of the CFD model.

The first can be done by defining more than one control points to morph the geometry or by using the Free Form Deformation approach. In this way it will be possible to represent all the possible shapes of the bulb. Important advantages can be brought by the use of mesh morphing techniques, which, if a lot of geometries are involved, saves time avoiding several mesh generations.

The second should involve an improvement of the OpenFOAM dictionaries, by investigating the effects of numerical schemes and refining the mesh. Furthermore, a more complete analysis should involve the dynamic of the ship, switching to the unsteady 6-Dof solver `interDyMFoam`. This will compute also the effect of the bulb configuration on the position of the ship, but will burden the numerical procedure.

Instead, regarding the optimization part many options can be tried. For example more than two design variables can be defined in order to include more information in the optimization problem (e.g. other bulb parameters, speed, load and so on). In this fashion more sophisticated sampling techniques are suggested, for example the Latin Hypercubes approach. Furthermore design, economical or other kind of constraints can be added to the problem. Finally, where resources allow, a multi-objective optimization can be performed.

Bibliography

[1] Web page. `https://www.gnu.org/philosophy/free-sw.html`.

[2] Web page. `https://cfd.direct/`.

[3] Web page. `http://www.optimad.it/products/mimmo`.

[4] Web page. `https://dakota.sandia.gov/`.

[5] Web page. `http://www.maersktechnology.com/en/all-stories/nose-job-for-better-efficiency`.

[6] Web page. `https://github.com/OpenFOAM/OpenFOAM-4.x/tree/master/tutorials/multiphase/interFoam/ras/DTCHull`,.

[7] 26th ITTC Specialist Committee on CFD in Marine Hydrodynamics, editor. *Practical Guidelines for Ship CFD Applications*, Rio de Janeiro, 2011.

[8] 27th ITTC Specialist Committee on CFD in Marine Hydrodynamics, editor. *Practical Guidelines for Ship Resistance CFD*, Copenhagen, 2014.

[9] A. Chiarini A. Scardigli, R. Arpa and H. Telib. Enabling of large scale aerodynamic shape optimization through pod-based reduced-order modeling and free form deformation. In *EUROGEN 2015*.

[10] D.A. Hudson A.F. Molland, S.R. Turnock. *Ship Resistance and Propulsion.* Cambridge University Press, 2011.

[11] A.J. Keane A.I.J. Forrester, A. Sobester. *Engineering Design via Surrogate Modeling: a Practical Guide.* John Wiley & Sons Ltd., 2008.

[12] M. S. Eldred G. Geraci J. D. Jakeman K. A. Maupin J. A. Monschke L. P. Swiler J. A. Stephens D. M. Vigil B. M. Adams, M. S. Ebeida and T. M. Wildey. *Dakota, A Multilevel Parallel Object-Oriented Framework for Design*

Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.5's User Manual, July 2014. Updated November 2016.

[13] S.H. Zak E.K.P. Chong. *An Introduction to Optimization*. Wiley-Interscience, 2001.

[14] M. Darwish F. Moukalled, L. Mangani. *The Finite Volume Method in Computational Fluid Dynamics*. Springer, 2016.

[15] Larrie D. Ferreiro. The social history of the bulbous bow. In Johns Hopkins University Press, editor, *Technology and Culture*, volume 52, pages 335–359. April 2011.

[16] Free Software Foundation. Gnu general public license v3. `http://www.gnu.org/licenses/gpl.html`, 2007.

[17] J. Guerrero. Design of experiments, space exploration, and numerical optimization using dakota and openfoam. Wolf Dynamics srl Advanced OpenFoam Course, February 2017.

[18] P. Van Oossanen J.D. Van Manen. *Principles of Naval Architecture*, volume II. The Society of Naval Architects and Marine Engineers, 1988.

[19] M. Wardetzky K. Crane, C. Weischedel. Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Trans. Graph. 28*, 2013.

[20] A.M. Kracht. Design of bulbous bow. In *SNAME Transactions*, volume 86, pages 197–217. 1978.

[21] J. Kent Layton. *Transatlantic Liners*. Bloomsbury Publishin, 2012.

[22] J.M. McFarland. *Uncertainty Analysis for Computer Simulations through Validation and Calibration*. PhD thesis, Vanderbilt Univesity, 2008.

[23] F.R. Menter. Two-equation eddy-viscosity turbulence models for engineering applications. *AIAA Journal*, 32(8), August 1994.

[24] M. F. Trujillo S. S. Deshpande, L. Anumolu. Evaluating the performance of the two-phase flow solver interfoam. *Computational Science & Discovery*, 2012.

[25] E. Tupper. *Introduction to Naval Architecture*. Butterworth-Heinemann, third edition, 1996.

[26] D.C. Wilcox. *Turbulence Modeling for CFD.* DCW Industries, 2006.